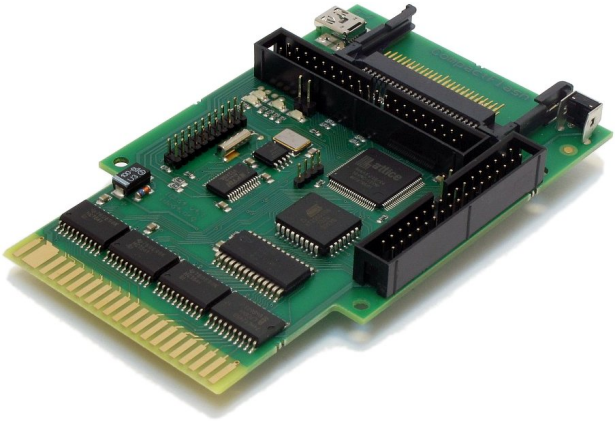
The IDE64 Project

IDE64
INTERFACE CARTRIDGE

user's guide

May 22, 2015



for card versions V2.1, V3.1, V3.2, V3.4, V3.4+, V4.1 and V4.2
with **IDEDOS 0.91 BETA 849**

THE ATA/ATAPI CONTROLLER CARD FOR COMMODORE 64/128 COMPUTERS
SUPPORTING HARD DISK, CDROM, DVD, ZIP DRIVE, LS-120 (A-DRIVE), COMPACTFLASH AND MORE

Document maintained by:

Kajtár Zsolt
Szigliget
Hóvirág u.15.
8264
Hungary
mail: soci at c64.rulez.org

Latest version of this document at: http://idedos.ide64.org/

Copyright © 2003–2015 Kajtár Zsolt (Soci/Singular).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “21 GNU Free Documentation License”.

Foreword

This is the official user's guide for the IDE64 interface cartridge V2.1, V3.1, V3.2, V3.4, V3.4+, V4.1 and V4.2 with IDEDOS 0.91. Incomplete but planned parts are marked ~~this way~~.

This document always represents the actual state of development and the facts stated here may or may not apply to future or old versions of IDEDOS or the IDE64 cartridge. *Please make sure you have the current version for your software and hardware!*

It's recommended that you read all sections of this manual. For most of your questions the answers are somewhere in this text. ;-)

Disclaimer

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies that could be damaging to your system, although any damage is highly unlikely. Proceed with caution; the author(s) do not take any responsibility.

Contents

1 About the cartridge 11

2 Hardware setup 15
2.1 Cabling, jumpers 15
2.2 Power supply 16
2.3 CompactFlash connector 16
2.4 Zip drive 17
2.5 LS-120, A-Drive 18
2.6 Peripherals 18
2.7 Battery 19
2.8 Let's start 19
2.9 Troubleshooting 20

3 The Setup utility 23
3.1 Standard setup 23
3.2 Color setup 28
3.3 Device numbers 29
3.4 ATA devices 31

4 Preparing a blank disk 35
4.1 The CFSfdisk utility 36
4.2 CFSfdisk notes 40
4.3 Mixed disks 41

5 Using partitions 43

6 Using directories **45**
6.1 Paths 48
6.2 Wildcards 50
6.3 Raw directory access 52

7 Using files **53**
7.1 SAVE 55
7.2 LOAD, VERIFY 56
7.3 OPEN 57
7.4 CLOSE 65
7.5 INPUT#, GET# 65
7.6 PRINT# 65
7.7 CMD 66
7.8 File operations 66

8 Direct access **67**
8.1 Block-read 67
8.2 Block-write 68
8.3 Buffer-pointer 69
8.4 TOC-read 70
8.5 Sub-channel-read 70

9 The File Manager **75**
9.1 Plugins 78
9.2 Manager configuration file 84

10 Using the monitor **89**
10.1 Starting the monitor 89
10.2 Disk commands 90

10.3 Display and modify memory 95
10.4 Execution control 102
10.5 Memory area commands 103
10.6 Miscellaneous 105

11 DOS Wedge **113**
11.1 At sign – DOS command 113
11.2 At, number sign – select device 113
11.3 At, dollar sign – list directory 114
11.4 Slash – load BASIC program 114
11.5 Percent sign – load assembly program 114
11.6 Apostrophe – verify assembly file 115
11.7 Up arrow – autostart BASIC program 115
11.8 Left arrow – save BASIC program 116
11.9 Pound sign – autostart assemble program 116
11.10 Period – change directory 117
11.11 Hashmark – execute shell 117

12 BASIC extensions **119**
12.1 CD – change directory 119
12.2 CDCLOSE – insert medium 120
12.3 CDOPEN – eject medium 120
12.4 CHANGE – change device number 120
12.5 DATE – display date 121
12.6 DEF – redefine F-keys 122
12.7 DIR – list directory 122
12.8 HDINIT – redetect drives 123
12.9 INIT – init memory 123
12.10 KILL – disable cartridge 124

12.11 KILLNEW – recover basic program 124
12.12 LL – long directory list 125
12.13 LOAD – load a program 127
12.14 MAN – start manager 127
12.15 MKDIR – create directory 128
12.16 RM – remove file 128
12.17 RMDIR – remove directory 128
12.18 SAVE – save a program 129
12.19 SYS – start ML program 129
12.20 VERIFY – verify program 130

13 Programming in assembly **133**
13.1 Standard KERNAL routines 133
13.2 IDE64 specific routines 151
13.3 IDE64 compatible programming 159

14 PCLink **171**
14.1 PCLink over IEC bus 171
14.2 PCLink over PC64 cable 172
14.3 PCLink over RS-232C 173
14.4 PCLink over ethernet 173
14.5 PCLink over USB 174

15 Command channel **175**
15.1 File management commands 175
15.2 Filesystem management commands 177
15.3 Partition management commands 181
15.4 Device management commands 183
15.5 Direct access commands 189

15.6 Directory handling commands 190
15.7 CD-ROM related commands 193
15.8 Misc commands 201

16 IDEDOS error messages 203

17 Compatibility 209
17.1 Hardware 209
17.2 Software 213

18 Updating IDEDOS 215

19 Filesystem checking 221
19.1 Using CFSfsck 221
19.2 Errors and resolutions 222

20 Frequently Asked Questions 225

21 GNU FDL 227

A The ShortBus 237

B The clock-port 243

C More information 247
C.1 Related Internet sites 247
C.2 Distributors 247

D Acronyms 249

1 About the cartridge

The IDE64 cartridge was created to provide the fastest I/O and biggest storage capacity available for the Commodore 64 and 128 computers.

With this cartridge it's possible to connect and use ATA(PI) drives like hard disks, CD-ROM and DVD drives, CompactFlash cards, Zip drive, LS-120 (A-Drive) or a networked host computer just like ordinary disk drives.

The cartridge contains a 64 or 128 KiB PEROM, which holds the IDEDOS disk operating system, a machine code monitor, file manager and setup utility. 28 KiB of RAM is used for internal buffers, and a battery powered real time clock chip is used for time keeping.

Two LEDs indicate the presence of cartridge and drive activity. A port called ShortBus is installed for peripherals like DUART Card, ETH64. There's also a RESET button for quick restarts.

The V4.1 cartridge adds a port to support clock-port peripherals (like ETH64 II, RR-Net), an USB FIFO chip for PCLink file transfers and two additional LEDs for secondary interface and PCLink activity. The V4.2 cartridge increases the PEROM size to 512 KiB.

The IDE64 cartridge is compatible with a wide variety of hardware including (but not limited to):

- Commodore serial bus drives, datassette
- CMD SuperCPU
- CMD FD-2000/FD-4000/HD
- JiffyDOS, Dolphin DOS
- PAL/NTSC C64 or C128 in C64 mode
- REU

- Turbo232, (E)TFE, RR-Net
- +60K
- 2nd SID

There are of course incompatible or unsupported hardware (most notably cracking or fastload cartridges, RamLink, etc.), for more information read section “17 Compatibility” at the end of the guide.

The architecture of the cartridge allows easy update of firmware, so it’s possible to be up to date with the latest versions of IDEDOS.

IDEDOS can handle four disks each clipped to a maximal 137 GB (128 GiB), and DVD drives up to 550 GB (512 GiB). A disk can be divided into a maximal of 16 partitions.

Files can be organized into a tree structured directory, each directory can contain 1023 files. Maximal file size is 4 GiB for regular files and 16 MiB for relative files. Filenames can be 16 characters long plus a 3 character file type. Automatic file timestamping is supported.

IDEDOS 0.91 is free software, the source and the tools required to build the firmware are public and available for POSIX systems and Win32. The source is licensed under the GPL-2:

> IDEDOS is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> IDEDOS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even

> the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

You are welcome to review the code and send suggestions, improvements or bug fixes if you want.

Please do not work around possible IDEDOS bugs in your software, better report them, so they can be fixed in future versions! Thanks.

2 Hardware setup

2.1 Cabling, jumpers

Figure 1: IDE64 V3.4 cartridge with a 128 MB CompactFlash card

Make sure that the computer is *powered off*. The cartridge must be plugged into the expansion port so that the chips are on top as seen on the picture.

Select the IDEDOS bank (on a V3.4+ cartridge) *while the computer is switched off*.

The USB PCLink cable of the V4.1 and V4.2 cartridge *must not be connected* to the host pc when plugging in the cartridge!

If using ethernet or serial PCLink, leave the network or serial cable disconnected. Connecting PC64 PCLink cable while either the host or the computer is powered is not recommened.

The ATA(PI) drives are connected with a 40 or 80 conductor IDE-cable to the port at the end of the cartridge. Make sure that the red line on the cable is at pin 1 on both ends. (pin 1 is usually near the RESET button at the right end of the cartridge, and near the power cord on the drives)

Two drives can be connected on the cable, one is called master, the other slave. It's important that there's only one master and one slave on a cable, so check the jumper setting of your drives. Single units must be configured as master, otherwise they might not be detected on boot.

2.2 Power supply

For reliable operation it's recommended to have a "heavy duty" power supply for the computer, as the original supplies (now probably over 20 years old) could be not up to the task of powering the cartridge and the attached peripherals. These supplies are mostly refurbished pc power supplies and can be found on the Internet with a little searching.

If only the integrated CF connector is used and no external devices are connected then no extra power supply is required, the slot is powered from the cartridge port.

External IDE-bus drives need external power too, so connect them to a pc power supply or to the matching connector of a heavy duty supply.

Unmodified pc AT-style power supplies will work without problems (some require a minimal load to start, so it may not work alone without any drives).

When using a ATX-style supply, these require that the green wire is connected to a black one for startup. Check the wire description on the box and do this at your own risk without connecting any drives to the supply to minimize possible damage. If everything goes well, the cooler must spin up.

First test your drives with the supply without connecting to the cartridge to see if everything is OK.

2.3 CompactFlash connector

The integrated CF connector on V3.4, V3.4+, V4.1 and V4.2 versions of the cartridge is a standard CompactFlash Type I connector. Cards

plugged into it don't need any external power supply.

Older cartridges can be equiped with an external IDE-CF adapter, these adapters need an external +5 V supply.

Cheap CF-MMC/SD/SDHC adapters can also be used instead of a real CF card. Some models are CompactFlash Type II and need slight "adjustment" of the connector to fit.

WARNING!

The integrated and external IDE-CF adapters are not hot plug capable, so never change the CF card while the computer or the slave drive is powered on!

If you want to use an ATA(PI) drive and CompactFlash card at the same time with V3.4 or V3.4+, then configure the drive as slave, because the CF card is always master.

With the V4.1 or V4.2 cartridge you can freely use two additional drives beside the CompactFlash card, as the CF connector is on a separate bus.

When using the on board CompactFlash connector and a slave drive with a 80 conductor cable, make sure that the IDE64 cartridge and the slave drive are connected to the drive connectors, while the board connector with the longer part of the cable remains unconnected! (The required PDIAG signal is not connected to the board connector on 80 conductor cables, while on 40 conductor cables it is)

2.4 Zip drive

When using an ATAPI Zip drive it's important that it's jumpered as Master A or Slave A, otherwise it won't work. (old drives without A

marking emulate a hard disk and do not need any special treatment)

NOTE

Not every Zip drive, cable and cartridge combinations work. It may be necessary to put the drive at the middle of the cable so that the remaining unconnected part of the cable is at least 15 cm long.

2.5 LS-120, A-Drive

The LS-120 (A-Drive) works fine with regular 720 kB, 1.2 MB and 1.44 MB floppy disks or with the 120 MiB SuperDisk. Disks of CBM 1581, CMD FD-2000 and CMD FD-4000 are not supported by the drive.

WARNING!

Read the ‘‘3.4.7 Linear write max’’ section before using LS-120 drives to avoid data corruption with some drive versions!

2.6 Peripherals

Peripherals (DUART card, ETH64, etc.) are connected to a 34 pin port (called ShortBus) of the cartridge with a 34 wire cable similar to the floppy cable in pcs, but without wire swapping. On the V4.1 and V4.2 cartridge there's a 22 pin clock-port for ETH64 II, RR-Net and other clock-port peripherals. Peripherals do not require any external power.

WARNING!

Never connect or disconnect IDE64 peripherals while the computer is powered on!

Read the "A The ShortBus" and "B The clock-port" appendixes for configuration information before using peripherals.

2.7 Battery

There's a battery holder on the back of the cartridge. It's strongly recommended to put a battery into it (CR2032 3 V) otherwise the setup settings can't be permanently stored and the file timestamping won't work. If using accu or super cap instead of battery, then enable recharging in the setup utility.

2.8 Let's start

When everything is ready, you may turn on your equipment. It's recommended to first turn on the power supply, then the computer, however if you have drives that do not spin-up until the computer is turned on and your AT-supply does not start without load (giving an annoying noise) you must do this the other way around.

If your drives are not detected on power on, try giving HDINIT on the BASIC prompt. If still nothing, turn off your computer and power supply, then check the cables and jumper settings. Sometimes changing the master or slave configuration or the drives might help.

If everything is OK the boot screen should come up first (black screen with light-blue characters) with the information on the version

of IDEDOS and the connected drives. Then the standard C64 reset screen should appear in less than 30 s, depending on the connected drives. Holding down CONTROL will hold the boot up screen, so it can be read. The boot screen only appears on power on.

2.9 Troubleshooting

On every power on IDEDOS does a short selftest of the hardware. There's a bit longer built in self test of IDEDOS when you hold down LEFTSHIFT while turning on the computer (or simply use SHIFT-LOCK).

NO PROPER BOOT Black screen, the red "stack" screen or garbled text colors on boot indicate a probably weak power supply. See section "2.2 Power supply".

IDEDOS V0.91 BETA 849 Everything looks fine. If you want to re-detect your drives in case some are missing then use HDINIT. If still nothing check the master/slave jumpers on drives and cabling. Also you might try to use a different combination of drives.

30719 BASIC BYTES FREE The firmware is likely missing or corrupted, try to update[1] it. Also this might indicate a broken PEROM, reseat it or install a new one.

CPU You are using the wrong version of IDEDOS, SuperCPU version on C64, or C64 version on SuperCPU.

[1] See section "18 Updating IDEDOS".

DE32 You have an old V2.1 cartridge, and tried to use a firmware compiled for a newer cartridge. Recompile IDEDOS for early V2.1, and update the firmware. Also this will happen when using the wrong firmware on the V4.1 or V4.2 cartridge.

ROM The IDEDOS ROM checksum does not match. Try to update the firmware. Clean contact edges, remove other cartidges or the whole port expander. Too long ShortBus cable might also cause this. Also you might try to reseat or replace the PEROM chip.

RAM The cartridge RAM buffer does not work reliable. Clean contact edges, remove other cartidges or the whole port expander. Too long ShortBus cable might also cause this. Also you might try to reseat or replace the RAM chip.

3 The Setup utility

The IDE64 cartridge has a battery backed up real time clock with some memory (DS1302) used to store the configuration settings and current time. The cartridge of course works without a battery too, but then it's using the default settings and no clock. If you do not like these defaults, you can modify them in the Setup utility, and then save them.

The Setup utility is started by pressing ← + RESTORE while in interactive mode. (like STOP + RESTORE) Moving is done with cursor keys, RETURN selects or changes item, + and − also changes item. C= and STOP exits sub menu, while C= saves setting and STOP discards them in main menu.

When the Setup utility is running it does not touch memory range $0800–$FFFF so your work won't lost when need to change some settings.

3.1 Standard setup

General settings. "CPULT" is the processor port leak time in 0.1 s resolution, depends on temperature and processor type.

3.1.1 Date, Time

Here's possible to set the built in clock. This has affect on the DATE command, the file timestamping and on TI$ (see "3.1.4 Set BASIC clock"). The calendar is built in and works in range 1980–2079. (Y2K compatible) The clock ticks while the computer is turned off. (but requires battery)

Figure 2: Start screen of the setup utility

3.1.2 Start boot file

The file called '1//:BOOT,PRG' can be auto started from the system drive at power up or always after reset. The file must be executable with RUN. Hold C= if you want to skip auto boot, or hold STOP to skip starting of program. The BASIC variable ST (at $90) contains 0 after power on, and 1 after reset.

3.1.3 Floppy speeder

Enables fast loader and saver for the 1541, 1570, 1571 and 1581 floppy drives, and fast read and write in manager. The floppy speeder is automatically disabled for unsupported drives, or if the drive is capable of the JiffyDOS fast protocol. In the later case the loading is

IDE64 IDEDOS v0.91
CMOS SETUP UTILITY
Date (mm:dd:yy) : Mon, Dec 23 2013
Time (hh:mm:ss) : 17 : 31 : 59
Start boot file : Disabled CPULT: 4
Floppy speeder : Enabled
Set basic clock : Disabled
Keyboard repeat : Disabled
Lowercase chars : Disabled
Use DOS wedge : Enabled
C128 keyboard : Disabled
Direct write : Disabled
Accu charging : Battery
CMD emulation : Disabled
Function keys : Enabled
Control stops : Disabled
Blocks free hack: Disabled
Fast detection : Disabled
ShortBus I/O 2 : Disabled
Stop Quit, C= Save, +/-, Return, CRSR

Figure 3: Standard setup screen of the setup utility

accelerated by the JiffyDOS protocol instead. Unfortunately some versions of 64HDD won't work unless this option is turned manually off. (For more read section "17 Compatibility"!)

3.1.4 Set BASIC clock

Sets the BASIC variable `TI$` to the time in the built in clock, after reset.

3.1.5 Keyboard repeat

Sets keyboard repeat after reset. (at `$028A`)

3.1.6 Lowercase chars

Select lower and uppercase chars or uppercase and graphics font after reset or STOP + RESTORE.

3.1.7 Use DOS wedge

IDEDOS overrides KERNAL based DOS wedges, unless it's disabled here. For more read sections "11 DOS Wedge" and "17 Compatibility"!

3.1.8 C128 keyboard

If cartridge has cartconfig register (some V2.1, and any later version of the cartridge), support is compiled in, and the computer is a C128 (without SuperCPU), it makes use of extra keys on keyboard. For list of keys see Table 1!

Key	Function
ESC	`CHR$(27)`
TAB	`CHR$(9)`
ALT	Nothing
HELP	Enter monitor
LINE FEED	`CHR$(10)`
NO SCROLL	Nothing
CURSOR KEYS and NUM PAD	Their usual meaning

Table 1: C128 extra keys

3.1.9 Direct write

Enables or disables the use of 'B=W'. Cleared on reset. You may have to enable it when creating filesystem.

3.1.10 Accu charging

Selects accu charging current. Leave it on Battery if not using accu or super cap. Charging a battery is useless, and some low quality products may even leak if charged.

3.1.11 CMD emulation

If enabled the returned ROM string for memory-read is different, and '/' is allowed in filenames which makes a difference in path handling. Also the partition listing is changed too.

3.1.12 Function keys

Enables function key support in direct mode. For list of default function key assignment see Table 15!

3.1.13 Control stops

If enabled, scrolling will be stopped, instead of slowed down, when CONTROL is pressed. It may be easier to find things on the screen, when it's not moving.

3.1.14 Blocks free hack

For compatibility with some BBS software the 'BLOCKS USED.' message can be changed to '32767 BLOCKS FREE.' at the end of directory listings. The problem can be noticed when a program refuses to save stuff because it thinks the disk is full ('0 BLOCKS USED.'), while actually the directory is empty, and there's plenty of space.

3.1.15 Fast detection

In some configurations IDEDOS can wait half a minute or more on system startup, while it's trying to detect some non-existing devices. This option will reduce the wait, however it might miss some slow starting devices.

3.1.16 ShortBus I/O 2

The V4.1 and V4.2 cartridge has integrated bus drivers for the ShortBus to improve reliability when using long ribbon cables.

If a port expander is used and there's some other hardware configured to be in the I/O_2 area then there will be a bus collision even if there's nothing connected to the ShortBus. Therefore this option is disabled by default.

If you're sure that there will be no bus conflicts and want to use ShortBus devices in the I/O_2 area then enable this option.

3.2 Color setup

The default colors of boot screen, manager and monitor can be changed here.

IDE64 IDEDOS v0.91
CMOS SETUP UTILITY
System drive : 12
Serial drive 8 : 8
Serial drive 9 : 9
Serial drive 10 : --
Serial drive 11 : --
IDEDOS drive 1 : 12
IDEDOS drive 2 : 13
IDEDOS drive 3 : 14
IDEDOS drive 4 : 10
IDEDOS drive 5 : 11
Network address : 0
Stop Quit, C= Save, +/-, Return, CRSR

Figure 4: Device number setup screen of the setup utility

3.3 Device numbers

Device number mapping to drives. −− means drive is not accessible.

NOTE

Choosing a device number from 15 to 17 can lead to unexpected incompatibilities with badly written software!

3.3.1 System drive

This device number is used for loading the DOS wedge shell command[2], auto booting[3], manager configuration file[4], and as the default

[2] 11.11 Hashmark – execute shell
[3] 3.1.2 Start boot file
[4] 9.2 Manager configuration file

work drive for the monitor[5]. Also this device will be selected (at $BA) after reset as the last used device for the DOS wedge commands[6].

3.3.2 Serial drive 8–11

Device number mapping for serial drives. Selecting 9 for serial drive 8, and 8 for serial drive 9 will effectively swap them. (of course only if not using direct serial routines)

3.3.3 IDEDOS drive 1–5

Device number mapping for IDEDOS drives. Drives are numbered continuously in the order of detection[7]. For a temporary device number change to device 8 it's simpler and faster to use the CHANGE command. See section "12 BASIC extensions"!

3.3.4 Network address

Used for EPCLink, it allows to use several C64 and hosts to share the same ethernet network. The network address setting must match on the C64 and host side for a successful communication. For simple setups leave it on 0.

[5] 10.2.1 Select work drive
[6] 11 DOS Wedge
[7] Check the output of HDINIT BASIC command for the order of drives.

IDE64 IDEDOS v0.91
CMOS SETUP UTILITY
Primary interface Master Slave
Power management: Default Default
Retry on error : Enabled Enabled
Write cache : Disabled Disabled
Read look-ahead : Disabled Disabled
Slow down CDROM : Enabled Enabled
Linear read max: 128 128
Linear write max: 128 128
Second. interface Master Slave
Power management: Default Default
Retry on error : Enabled Enabled
Write cache : Disabled Disabled
Read look-ahead : Disabled Disabled
Slow down CDROM : Enabled Enabled
Linear read max: 128 128
Linear write max: 128 128
Stop Quit, C= Save, +/-, Return, CRSR

Figure 5: ATA devices screen of the setup utility

3.4 ATA devices

Various device "features" can be configured here for master and slave ATA(PI) drives on primary and secondary interfaces. Some of these settings affect the performance of IDEDOS.

3.4.1 Power management

Drive spin down time from disabled to 2 hours in 15 steps. Default means do not touch drive's power management settings. (almost all drives support power management) The KILL! command will only spin down those drives which have power management enabled here.

3.4.2 Retry on error

Tells the drive to retry on error.

3.4.3 Write cache

Will speed up writes if supported by the drive.

3.4.4 Read look-ahead

Will speed up short sequential reads if supported by the drive. If you set "Linear read max" to it's maximum, this does not affect performance significantly.

WARNING!

Do not enable ''Read look-ahead'' on the ancient LPS170A harddrive, it's bugged firmware will mix up sectors sometimes, and it'll damage the filesystem. Modern drives should be OK.

3.4.5 Slow down CD-ROM

Selects 1× speed on CD-ROMs and DVDs supporting speed settings. This reduces noise of faster than 24× drives and gives faster spin up times. Access times will be a bit longer, and small cache drives will copy tons of small files also a bit longer. This option has no effect on the bulk transfer rate on stock C64, because the cartridge transfers data much slower, however on SuperCPU this may limit reading speed. Also if you experience unreliable behaviour from the drive (copying lot of files fail at different places due to constant spin up and downs) then disable this.

3.4.6 Linear read max

Maximal sequential read allowed by the drive. (setting it lower than 128 will degrade the performance of CFS filesystem greatly!) You should decrease it from 128 if you get `?LOAD ERROR` when loading a big program. (this is very unlikely)

3.4.7 Linear write max

Same as above but for write operations.

WARNING!

Some versions of LS-120 drives with 1.2 MB or 1.44 MB disk corrupt files if ‘‘Linear write max’’ is not set less than 12 (e.g. 11)! It’s dependent on the drive’s firmware version and/or producer, so test your drive first! (With ZIP file extraction, 224 blocks program copy and load, etc.)

4 Preparing a blank disk

To use a new disk with IDE64 you must first create the filesystem on it. IDEDOS has it's own filesystem called CFS. It allows to use disks up to 128 GiB, files up to 4 GiB with holes and fast seeking, relative files up to 16 MiB, 16 partitions, nested directories, customizable file types, and new file permissions.

To create the filesystem on a hard drive, CompactFlash, Zip drive and LS-120 drive use the provided CFSfdisk utility. It allows to create partitions, change partition flags and create filesystems on them.

NOTE

You have to repartition the medium to IDEDOS's native filesystem in order to store data on them! Many new disks come with filesystems not suitable for CBM files (e.g. FAT, NTFS).

First set the "Direct write" option[8] in the setup utility to enabled, then start the format utility. Follow the instructions. Don't forget that formatting a partition will erase all data on it permanently!

CD-ROM/DVD drives do not need any special treatment, just use ISO9660 or Joliet format CDs to be able to read them. (Rock Ridge Extensions are not supported, these CDs will have short filenames unless Joliet extension is also present) Multi-session and mixed format discs are both supported. Everything after the last dot or after the last comma will be used as file type. There's a compile time option for disabling the automatic extension to file type conversion, in case you'd like to have everything as 'PRG'.

[8] See section "3.1.9 Direct write".

DVD drives work no different than CD-ROM drives, DVDs written with the ISO9660/Joilet format will work, and ISO9660+UDF (most DVDs) will only have short filenames, as the UDF filesystem is not supported.

Floppy disks for LS-120 may require physical format before the creation of the filesystem. Use the format (N) command described in section "15 Command channel".

4.1 The CFSfdisk utility

The partitioner tool and the CFS filesystem creator is integrated into one executable called CFSfdisk. First it was a prototype utility running on GNU/Linux systems, then it was ported to C64 with the CC65 compiler, and finally it was rewritten in assembly by hand. CFSfdisk is released under the GPL-2, the 64tass source can be downloaded with the IDEDOS source.

Before partitioning or formatting go into the setup and change "`Direct write`" to enabled. Then load and start CFSfdisk.

```
CFSfdisk version 11a
Copyright (C) 2001-2011 Kajtar Zsolt (Soci/Singular)

CFSfdisk comes with ABSOLUTELY NO WARRANTY; This is free
software, and you are welcome to redistribute it under
certain conditions; see LICENSE for details. (GPL-2)

Drive number (4-30, default 12): _
```

CFSfdisk first needs to know the device number of the drive to be repartitioned. It's usually 12 or 13, see your setup settings on device number assignment for drives. Remember that there will be *no* changes made on disk unless you write them with command 'W'.

```
Did not found any CFS partition entry in PC BIOS partition
table. I assume you want to use the whole disk.

If you want to share the disk with other operating systems, use
the fdisk utility, and make a partition entry with type 0xCF.

Creating new disklabel.
Creating new partition table.
```


All is good so far, the whole disk will be used for IDEDOS. If you only want to use some part of it instead, then read the notes at the end of this chapter.

```
Command (m for help): m
Command action
  a  select boot partition       o  clear partition table
  b  change partition's name     p  print the partition table
  c  clean boot sector           r  toggle writeable flag
  d  delete partition            s  save table to file
  f  toggle formatting flag      t  change partition's type
  g  set global disklabel        u  load backup partition table
  h  toggle hidden flag          q  quit without saving changes
  l  load table from file        w  write changes to disk
  m  print this menu             x  expert geometry setup
  n  add new partition
```

Typing '`M`' lists possibilities. There are no partitions yet, so let's create one.

```
Command (m for help): n
Partition number (1-16, default 1): 1
Start (2-1253951, default 2):
Use 123, +123, +123G, +123M or +123K
(9-1253951, default 1253951): +306m
Partition's name: stuff
```

This creates partition 1 called '`stuff`' beginning on the start of the disk, and it will fill the half of the disk. ($\sim$300 MiB) It's possible

to give the exact start and end position or the size of the partition in sectors (e.g. +2342) for power users. For everyday use +1048576K or +1024M or +1G creates an example partition with a size of 1 GiB. (metrics are powers of 1024, not 1000!)

After adding some more partitions, here's an example partition list:

```
Command (m for help): p

Drive 12: 1253952 sectors (612 MiB)
Disklabel: idedos disklabel

Nr Flags   Start       End Size(KiB) Id System  Name
 1*F--         3    626690    313344  1 CFS     stuff
 2 F--    626691    831490    102400  1 CFS     work
 3 F--    831491    864258     16384  1 CFS     incoming
 4 F--    864259   1253951    194846+ 1 CFS     backup
```

Partitions marked with the `F` flag will be formatted, `H` will be hidden and `R` will be read only. The asterix in front marks the boot partition.

Now let's change the global disklabel, and the default boot partition (I like to start at the work partition after boot).

```
Command (m for help): g
New disklabel: soci's disk

Command (m for help): a
Partition number (1-16, default 1): 2
Partition 1 set as boot partition.
```

That's all for now, let's start formatting (or you can use '`Q`' to abort):

```
Command (m for help): w
```

```
Drive 12: 1253952 sectors (612 MiB)
Disklabel: soci's disk

Nr Flags   Start       End Size(KiB) Id System  Name
 1 F--         3    626690    313344  1 CFS     stuff
 2*F--    626691    831490    102400  1 CFS     work
 3 F--    831491    864258     16384  1 CFS     incoming
 4 F--    864259   1253951    194846+ 1 CFS     backup

Write out this partition table (Y/N, default N): y
Formatting partition 1...done.
Formatting partition 2...done.
Formatting partition 3...done.
Formatting partition 4...done.


*** TURN OFF THE COMPUTER WHEN FINISHED PARTITIONING ***
```

The maximal formatting time of a 137 GB partition on a harddisk takes ∼28 min, this is slightly faster on CF cards. There's a countdown displayed while the formatting is in progress. Big partitions like this are not recommended, as checking the integrity with CFSfsck even if it's empty takes at least twice as long.

After filling partition 4 with lot of important stuff, let's mark it read only. Load the CFSfdisk utility and start it.

```
Command (m for help): r
Partition number (1-16, default 1): 4
Partition 4 flags toggled.

Command (m for help): w

Drive 12: 1253952 sectors (612 MiB)
Disklabel: soci's disk

Nr Flags   Start       End Size(KiB) Id System  Name
```

```
 1 ---         3    626690    313344  1 CFS     stuff
 2*---    626691    831490    102400  1 CFS     work
 3 ---    831491    864258     16384  1 CFS     incoming
 4 --R    864259   1253951    194846+ 1 CFS     backup

Write out this partition table (Y/N, default N): y


*** TURN OFF THE COMPUTER WHEN FINISHED PARTITIONING ***
```

Hiding a partition can be done the same way, by using the ‘H’ command.

4.2 CFSfdisk notes

CFSfdisk auto detects CHS and LBA disks, so you don’t have to worry about this. But if it’s wrong and you want to change it, use “expert geometry setup”. The current setting is visible in the ‘Drive xx:’ line, if there are numbers about cylinders, etc. then the disk is in CHS format.

If you only want to reformat a partition without deleting and re-adding it, just use the “toggle formatting flag” command to toggle the format flag. Or change it’s type to CFS even if it was already in CFS format, by using the “change partition’s type” command.

The “clean boot sector” command can be used to force CFSfdisk to treat the disk as new. This can be used to ignore the CF BIOS partition setting, if there’s any.

The “clear partition table” command will remove all partitions in one command, while the “delete partition” command just removes the selected one.

It’s possible to recover a deleted partition by adding it as new with

the known correct start and end addresses, and then switching off the formatting flag for the partition before writing changes to disk.

Partition layout can be backed up and restored by using the commands “load table from file” and “save table to file”. CFSfdisk checks if the file is for this device or not, to avoid mistakes. Use the “toggle formatting flag” to reformat partitions as needed. By default none of the partitions will be formatted after loading a backup.

By starting with CFSfdisk version 10, the backup partition table is created at the end of the disk, unless the location was already set by ealier partitioning. The backup table is automatically syncronized with the primary table on writing changes to disk, or when using CFSfsck. The “load backup partition table” can be used to load it, in case of emergency. This function does not work well with disks partitioned by earlier versions of CFSfdisk when the the boot sector was lost, as then the guess for the location will be wrong.

4.3 Mixed disks

It’s possible to mix CFS partitions with other non-native partitions like FAT when using the MSDOS partitioning sheme.

To do this create a primary partition with type `0xCF` by a partitioning tool of your choice, and rewrite or clear the MBR. This special partition will be recognized by CFSfdisk, and all partitions and partition tables for IDEDOS will be created within this area.

There’s even a custom MBR loader for x86 systems, which can live together with the IDEDOS signature in the boot sector, and boot other partitions.

Take care not to destroy the CFS signature in the boot sector when installing other systems!

FAT partitions in the MSDOS partition table will be recognized and combined with the CFS entries, if there's enough space in the partition table.

5 Using partitions

When using a new disk at least one partition must be created. Partitions provide the highest level of organizing data. Each partition has it's own filesystem, so possible disk or software errors can't destroy the whole data at once.

It's not recommended to create only one huge 80 GiB partition for all data (which won't be more than a few GiB I guess), because the filesystem checking of such a big partition will take a while... Also it's not necessary to partition all space on disk, as it's possible to create additional partitions later if needed.

One can select the default partition on boot, set the global disk label, partition names and partition attributes (hidden and read only) with the CFSfdisk utility. The read only attribute is useful to prevent accidental changes to the partition.

IDEDOS supports the MSDOS partitioning sheme as well, when it's looking for FAT filesystems on a disk. The FAT partitions show up as additional unnamed foreign partitions (FOR) after the normal CFS partitions. Because each of them are using 2 entries, only 8 FAT partitions are supported per drive. An ISO9660 or Joilet formatted media gets a single unnamed partition entry.

Examples:

Listing available partitions. '∗' indicates default partition, '<' indicates read only partition. Hidden partitions are not listed.

```
@$=P
255 "SOCI'S DISK      " IDE64
1     "STUFF"            CFS
2     "WORK"            *CFS
3     "INCOMING"         CFS
4     "BACKUP"           CFS<
5     "                " FOR<
```

```
5 PARTITIONS.
READY.
```

Selecting partition 4 as working partition:

```
@CP4
02, PARTITION SELECTED,004,000,000,000

READY.
```

Selecting partition 1 as working partition the other way:

```
open 15,12,15,"cP"+chr$(1):close15

ready.
```

Load a file from partition 2 from the directory '/GT'. More about paths in section "6.1 Paths".

```
LOAD"2//GT/:FILE"

SEARCHING FOR 2//GT/:FILE
LOADING $0801-$0932
READY.
```

6 Using directories

A directory is a list of files. To view the directory use LOAD"$" and LIST, DIR, or @$. Of course LOAD will overwrite the current program in memory, while the last 2 methods will preserve the computer's memory content.

The number before the first line is the partition number, it's followed by the directory label, and finally the ID string IDE64. The following lines provide information about the size of each file in 256 byte blocks (so 4 blocks are exactly 1 KiB), the name of the file enclosed in quotation marks, and it's file type at the right side. The last line is the used block count in the directory. This can be set to be always '32767 BLOCKS FREE.' for compatibility with certain software.

Example:

A simple directory list, using LOAD"$":

```
LIST

2 "TEST              " IDE64
0    "PLUGINS"          DIR
40   "BOOT"             PRG
2    "MAN"              USR
5    "TOD"              ASM
23   "VIEWER"           PRG<
70 BLOCKS USED.
READY.
```

It's possible to get a bit more detailed directory list, which is similar to the first one, but instead of the full filetype, only one letter is present. Then the timestamp of the file is displayed as month, day, hour, minute and the first letter of AM/PM.

Example:

Letter	Short	Filetype
-	DEL	Deleted entry
S	SEQ	Sequential file
P	PRG	Program file
U	USR	User file
R	REL	Relative file
D	DIR	Directory
L	LNK	Link
?		Other, user defined

Table 2: Detailed directory filetypes

A detailed directory list, using LOAD"$=T":

```
LIST

2 "TEST           " IDE64
0    "PLUGINS"          D 10/14 08.16 P
40   "BOOT"             P 11/11 07.39 P
2    "MAN"              U 11/15 09.48 A
5    "TOD"              ? 09/12 10.55 A
23   "VIEWER"           P 07/14 07.26 P
70 BLOCKS USED.
READY.
```

There's an even more verbose directory listing mode with dates, which includes the full filetype, the protection flag, and the modification year too.

Example:

A more detailed directory list, using LOAD"$=T*=L":

```
LIST

2 "TEST           " IDE64
0    "PLUGINS"          DIR   10/14/04
 08.16 PM
```

```
40   "BOOT"              PRG    11/11/04
 07.39 PM
2    "MAN"               USR    11/15/04
 09.48 AM
5    "TOD"               ASM    09/12/04
 10.55 AM
23   "VIEWER"            PRG<   07/14/04
 07.26 PM
70 BLOCKS USED.
READY.
```


Putting a few hundred files in one long directory is not an optimal way of organizing data, so IDEDOS provides subdirectories.

Subdirectories look like normal files with file type DIR in directory listings. Directories are organized into a tree like structure starting from the root directory.

The root directory is the top level directory, it's parent directory is itself. After boot this directory is selected as the working directory.

The working directory is the directory which is used when no path is given in the filename just like in LOAD"$".

Managing directories are done via channel #15 commands, but these examples will use the DOS Wedge to simplify things. (These commands are described in detail in section "15.6 Directory handling commands")

Examples:

Creating a subdirectory

```
@MD:DIRNAME
```

Changing the working directory

```
@CD:DIRNAME
```

Changing the working directory to the parent directory

```
@CD←
@CD:..
```

Changing the root directory

```
@CR:DIRNAME
```


Removing an empty subdirectory

```
@RD:DIRNAME
```

Changing the name of a subdirectory

```
@R:NEWNAME=OLDNAME
```

Moving a subdirectory tree into a different directory

```
@R/NEWPATH/:NEWNAME=/OLDPATH/:OLDNAME
```

Changing the directory's label

```
@R-H:NEWHEADER
```

Write protecting a directory against modifications

```
@L:DIRNAME
```

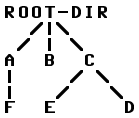
Hiding a directory

```
@EH:DIRNAME
```

6.1 Paths

Each file in the tree structure can be reached with a “path”. The path is composed from the name of the directories separated by a slash character.

Here’s an example directory structure:



Let’s say the working directory is ‘C’ now. To load a file from directory ‘E’ use the following: `LOAD"/E/:FILE"`. This is a relative path. It’s also possible to specify the location of a file from the root directory, it’s called absolute path: `LOAD"//C/E/:FILE"`. As it seems the path is enclosed between 2 slashes before a semicolon, after it is the name of file. In the last example the real path was ‘`/C/E`’ where the slash before ‘C’ indicates the it’s an absolute path.

Now let’s compose a relative path, but now our working directory is ‘F’: `LOAD"/../../C/E/:FILE"`. There’s no directory named ‘`..`’ in the graph, as this is a special directory, and it means the parent directory. That’s why ‘`CD:..`’ means change to parent directory in some previous example. There’s another special directory called ‘`.`’ which means the directory itself. These two special directories always exists.

To embed the partition number into the filename place it before the path, e.g. ‘`13/C/:E`’, and don’t forget to include the colon!

```
10 OPEN 15,12,15
20 PRINT#15,"CD//":REM GOTO ROOT-DIR
30 PRINT#15,"MD:A":REM CREATE A
```

```
40 PRINT#15,"MD/A/:F":REM CREATE F
50 PRINT#15,"MD:B":REM CREATE B
60 PRINT#15,"CD:B":REM ENTER B JUST FOR FUN
70 PRINT#15,"MD//:C":REM CREATE C
80 PRINT#15,"MD//C/:E":REM CREATE E
90 PRINT#15,"MD/../C/./E/../:D":REM CREATE D ;-)
100 CLOSE 15
```

Listing 1: The directory structure above can be created by this program

6.2 Wildcards

It's possible to filter directory listings to show only a subset of files in a directory by using wildcards. There are 2 different wildcards: '?' matches exactly one character, while '∗' matches any number of characters. These wildcards can be used in path elements or filename too, in this case the first filename will be matched. To filter directory listing by file type, append '=TYP' to the pattern. The following one character file type shortcuts are recognized: P→PRG, S→SEQ, U→USR, R→REL, J→LNK, D→DEL, B→DIR.

Wildcard filtering does only work on formatted directory lists (secondary address 0).

Examples:

List files starting with 'A'.

```
LOAD"$A*"

SEARCHING FOR $A*
LOADING $0801-$08DF
READY.
LIST

1 "PICTURES        " IDE64
132  "ALIEN BAR BY WEC" FUN
29   "ALIEN-LIFE BY MB" SHI
```

```
132  "ALLEY BY MIKE"     FUN
132  "ANGEL BY FZ"       FUN
72   "ARNIE BY AMN"      DRL
497 BLOCKS USED.
READY.
```

List files ending with ‘DEPACKER’.

```
@$*DEPACKER

1 "UTILITIES          " IDE64
9    "DMC 5.0 DEPACKER" PRG
10   "DMC 5.1 DEPACKER" PRG
19 BLOCKS USED.
READY.
```

List files containing ‘PLAYER’.

```
@$*PLAYER*

1 "UTILITIES          " IDE64
0    "MUSIC PLAYER"     DIR
9    "CD PLAYER 1"      PRG
12   "CD PLAYER 3"      PRG
12   "CD PLAYER 4"      PRG
77   "CUBIC PLAYER"     PRG
110 BLOCKS USED.
READY.
```

List 4 character long filenames.

```
@$????

1 "UTILITIES          " IDE64
0    "DEMO"             DIR
0 BLOCKS USED.
READY.
```

List files starting with ‘K’ and file type ‘D64’

```
@$K*=D64

1 "DEMOS              " IDE64
```

```
683  "KJU"               D64
683  "KRESTAGE 2"        D64
768  "KRESTOLOGY 1"      D64
768  "KRESTOLOGY 2"      D64
683  "KRESTOOLS"         D64
3585 BLOCKS USED.
READY.
```

6.3 Raw directory access

Ever wondered how to access creation date or attributes of files just like the BASIC command LL command does? It's done by using raw directory access!

Raw directory access gives low level access to the filesystems directory structure, so it's format is not uniform across drives.

To open a raw directory channel, use secondary address 2–14. The first character for IDE64 drives is always 'I' ($49). Then the 32 byte directory entries follow, in format of the CFS filesystem's directory entry layout. The end of list can be detected by the status variable. (6th bit set, End Of File)

7 Using files

Files store programs and data in a filesystem. Traditionally files could only be accessed sequentially like on tape, but later relative files appeared with fixed record lengths on disk drives.

Using IDEDOS with the CFS filesystem it's now possible to use randomly accessible files up to 4 GiB without the fixed record length limitation for both read and write at the same time.

Unlike other systems it's possible to seek beyond the end of a file when writing or modifying and create "holes", which are filled with nulls between the file's last end and the newly written data. These holes of course do not use any disk space. This concept also applies to relative files for it's unused records, but instead of nulls the empty record pattern is used.

Relative files are available for compatibility reasons, they are limited to the usual 65535 records of 255 bytes, nearly ∼16 MiB.

There's also a special purpose file called link, which can be used to reference other files. It contains a path to a new location.

IDEDOS has 10 buffers for it's own files, these are shared between the IDE64 drives.

Of course IDEDOS supports opening of multiple files on the same or different partitions simultaneously for writing and modifying. But keep in mind that each file locks a 2 MiB area of the partition it's located on, which means that on partitions smaller than 2 MiB (like on a 1.44 MB disk) there can be only one file opened for write at the same time, and the next file open for write, modify or create (including directories) will fail with a disk full error. File scratches or directory removes will work as these are handled specially in this case. You'll won't ever notice this disk area locking on a big partition except when

it's nearly full.

All open files are locked, so they cannot be moved across directories or removed until they are not closed. This applies to directories too, the current working directory cannot be removed even if it's empty. Opening the same file several times is not allowed as well.

When creating new files, the default attributes will be deletable, readable, writable, loadable, and not hidden, except for relative files, which will be non-loadable. The directory, partition and the disk must be writable where the file is created, and the file must be deletable if replaced.

Wildcards ('?' and '$*$') and special characters like '`:`', '`,`' and '$=$' are not allowed in filename when creating a new file or directory. The '`/`' is treated as a path separator for backwards compatibility, unless it's disabled in the setup, then it's just a regular character. The file type must be at least two characters long, with the same limitations on usable characters, plus it can't contain space, '>' and '<'. The following one character file type shortcuts are recognized: P→PRG, S→SEQ, U→USR, L→REL, J→LNK, D→DEL.

IDEDOS properly closes it's files when CLALL is called to reduce the chance of creating non-closed files. (this happens when issuing a RUN, CLR, NEW, or just entering a new BASIC line) Also non-closed files do not automatically mean lost sectors, unless huge amounts of data has been written to the file without closing it.

In the format descriptions everything between '[' and ']' is optional, and '⟨*name*⟩' means a parameter.

7.1 SAVE

The BASIC command SAVE (and the similar KERNAL call) has the following format:

Format:

SAVE"[[@][⟨*partition #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩]"[,⟨*device #*⟩[,⟨*mode #*⟩]]

The device number identifies the drive. There's a nice table at page 136 which lists valid device numbers. If no device number is given, then the last one is used ($BA).

The mode parameter does not matter, except for datassette, where bit 0 disables the relocation on loading, and bit 1 means write end of tape marker.

The file name and type can't contain any special characters. If no filetype is specified then 'PRG' is used. The specified file shouldn't exists, unless replace is used (@), in this case it's possible to use wildcards.

NOTE

Unlike other drives IDEDOS first removes the original file before replacing it.

Examples:

Simple save into current drive, current partition and into working directory, the result is 'A' with file type 'PRG'.

```
SAVE"A"
```

This will save file 'B' with file type 'DAT' on device 12 to partition 3 to directory '/A'. If it's exists it will be first removed.

```
SAVE"@3//A/:B,DAT",12
```

This will overwrite the first file in working directory beginning with 'A'. File type is not changed. Wildcards are only allowed if replace is used! ('@:')

```
SAVE"@:A*"
```

7.2 LOAD, VERIFY

The BASIC commands LOAD and VERIFY have the following format:

Format:

LOAD"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩]"[,⟨*device #*⟩[,⟨*mode #*⟩]]
VERIFY"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩]"[,⟨*device #*⟩[,⟨*mode #*⟩]]

The file must be loadable. (the executable flag must be set) For directory list load the directory must be readable. If no filetype is specified then '∗' is used.

The device number identifies the drive. If no device number is given, then the last one is used ($BA).

If mode=0 then load to BASIC program start, otherwise load to original address. If no mode is given, it's 0.

Examples:

Simple load from current drive, current partition and from working directory. As file type is not specified the first will be loaded. The starting address will be ignored. (no mode specified means mode=0)

```
LOAD"A"
```

This will load file 'B' with file type 'PIC' from device 13 from partition 4 from directory '/A'. It will be loaded to the load address included in file (first 2 bytes), because mode≠0.

```
LOAD"4//A/:B,PIC",13,1
```

This will load the directory listing containing only 'PRG' files as a BASIC program.

```
LOAD"$*=P",12
```

7.3 OPEN

Things are getting complicated here. The usual syntax of OPEN is as follows:

Format:

OPEN ⟨*file #*⟩[,⟨*device #*⟩[,⟨*channel #*⟩[,"⟨*text*⟩"]]]

The file number identifies a file for IDEDOS, it must be in the range 1–127. If it's 128–255 then BASIC adds a CHR$(10) after each line. (it's an extra linefeed for certain printers)

The device number identifies the drive. If no device number is given, then the last one is used ($BA).

The channel number identifies the communication channel for the drive assigned to a certain file, so it must be choosen unique per drive. The available channel numbers are described on page 137. Some commands use this channel number to identify the file. (like the position command, or direct access commands) If no channel number is given, then 0 is used for keyboard, tape, RS-232, and 255 for screen, serial and IDE64 drives.

The open formats presented here can be easily converted to assembly.

7.3.1 Opening a direct channel

The usage of direct channel is described in section “8 Direct access” in detail. The format of open is:

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"#"

Reading the last byte of the buffer will set the end of file bit in ST.

7.3.2 Opening a formatted directory list

Formatted directory list is a list of files in a directory formatted as a BASIC program.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,0,"$[⟨*partition #*⟩]"
OPEN ⟨*file #*⟩,⟨*device #*⟩,0,"$[[⟨*part #*⟩][⟨*path*⟩]:][⟨*pattern*⟩]"

Reading the last byte of the directory list will set the end of file bit in ST.

```
10 OPEN 2,12,0,"$":GET#2,A$,A$
20 GET#2,A$,A$:IF ST THEN CLOSE 2:END
30 GET#2,A$,B$:REM SIZE
40 PRINT ASC(A$+CHR$(0))+ASC(B$+CHR$(0))*256;
50 GET#2,A$:PRINT A$;:IF A$ THEN 50
60 PRINT:GOTO 20
```

Listing 2: This small program prints the directory list

7.3.3 Opening a raw directory list

Similar to the formatted directory, but gives all possible information about a file. It’s not drive independent.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"$"

Reading the last byte of the directory list will set the end of file bit in ST.

7.3.4 Opening a regular file for read

One way is to use secondary address 0, which means read only sequential access. If read only random access is required, then secondary address 2–14 must be used.

If read only random access is required, then the secondary address 2–14 must be used. The read operation is requested by the ',R' after the filetype or file name, it's usage is optional. If no filetype is specified then '∗' is used.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,0,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩]"
OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩][,R]"

The opened file have to be readable and must exists.

```
0 OPEN 15,12,15:CLOSE 15
5 IF ST THEN PRINT"DEVICE NOT PRESENT":END
7 OPEN 15,12,15
10 OPEN 2,12,0,"FILE,PRG":N$=CHR$(0)
20 GET#2,A$,B$:S=ST:REM SAVE STATUS
30 INPUT#15,A,A$,B,C,D,E:IF A=0 THEN 50
40 PRINT"DISK ERROR:"A;A$;B;C;D;E:GOTO 90
50 IF S=66 THEN PRINT"TOO SHORT!":GOTO 90
60 IF S=64 THEN PRINT"START ONLY?":GOTO 80
70 IF S THEN INPUT#15,A,A$,B,C,D,E:GOTO 40
80 PRINT"START:"ASC(A$+N$)+ASC(B$+N$)*256
```

```
90 CLOSE 2:CLOSE 15
```

Listing 3: This small program prints the starting address of a file, and has some nice error checking.

7.3.5 Creating or replacing a regular file

If only one output file with write only sequential access is required, then the simplest is to use secondary address 1. In this case if the filetype is not specified then 'PRG' is used.

If write only random access is required, then the secondary address 2–14 must be used. The write operation is requested by the ',W' after the filetype or file name. If the filetype is not specified then 'SEQ' is used.

Format:

OPEN ⟨*file #*⟩,⟨*dev #*⟩,1,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩]"
OPEN ⟨*file #*⟩,⟨*dev #*⟩,⟨*2–14*⟩,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩],W"

The file name and type can't contain any special characters. If no filetype is specified then 'SEQ' is used. The specified file shouldn't exists, unless replace is used (@), in this case it's possible to use wildcards.

NOTE

Unlike other drives IDEDOS first removes the original file before replacing it.

```
10 OPEN 2,12,1,"@:FILE,SEQ"
20 PRINT#2,"HELLO";
```

```
30 IF ST THEN PRINT"ERROR DURING WRITE"
40 CLOSE 2
```

Listing 4: This small program creates a sequential file and writes to it.

7.3.6 Opening a regular file for append

Appending to a file means write only random access starting from the end of the file. The append operation is requested by the ‘,A’ after the filetype or file name. If no filetype is specified then ‘∗’ is used.

Format:

OPEN ⟨*file #*⟩,⟨*dev #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩],A"

The file must be read and writable, and must exists.

7.3.7 Opening a regular file for read and write

If a file have to be both read and writable with random access use this opening mode. This operation is requested by the ‘,M’ after the filetype or file name. If no filetype is specified then ‘∗’ is used.

Format:

OPEN ⟨*file #*⟩,⟨*dev #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩[,⟨*type*⟩],M"

The file must be read and writable, and must exists.

7.3.8 Opening a relative file for read and write

For compatibility IDEDOS provides “relative files”. These are fixed record length read and writable files. The record length can be in range 1–255, and maximum of 65535 records are supported. For

opening relative files the filetype 'L' have to be used. When creating the file for the first time the record length must be added after the filetype, as shown below.

Format:

OPEN ⟨*file #*⟩,⟨*dev #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,L,"+
CHR$(⟨*record length #*⟩)
OPEN ⟨*file #*⟩,⟨*dev #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,L"

IDEDOS initializes all 65535 records with a CHR$(255) character on creation, but fortunately these records do not waste any disk space, as only those will be really allocated which are actually used. Non-IDE64 drives may not initialize all records, so it's not a bad idea to write a CHR$(255) into the highest record number when creating the file. Using a record size of 255 or 1 will most likely work only on IDEDOS, but nowhere else.

At most the "record size" number of characters can be written into a record, attempting to write more will result in an error message, and characters will be dropped. Calling CLRCHN after writing data will move to the next record. (this means everything have to be written in one PRINT# statement for BASIC programmers, as normally)

When reading back a record the end of file bit in ST will be set on the last character of a record. You have to call CLRCHN after EOF to move to the next record. This is automatic in BASIC.

Selecting a record is done through the command channel with the position command, this is described in section "15 Command channel" in detail.

Written records are cached in memory. Do not forget to select a record or close the file to force unwritten data to disk. This is also important for CBM and other drives!

```
10 INPUT"SOURCE DRIVE";S
20 INPUT"SOURCE NAME";S$
30 INPUT"DESTINATION DRIVE";D
40 INPUT"DESTINATION NAME";D$
50 OPEN 2,S,2,S$+",L":OPEN 15,S,15:A=128:R=0
60 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(R+A)
70 INPUT#15,C:IF C=0 THEN R=R+A
80 A=A/2:IF A>=1 THEN 60
90 OPEN 3,D,3,D$+",L,"+CHR$(R)
100 PRINT#15,"P"CHR$(2)CHR$(1)CHR$(0)CHR$(1)
110 A$=""
120 GET#2,B$:A$=A$+B$:IF ST=0 THEN 120
130 INPUT#15,C:IF C=0 THEN PRINT#3,A$;:GOTO 110
140 CLOSE 3:CLOSE 2:CLOSE 15
```

Listing 5: This small program copies a relative file. The record length and number of records are automatically detected.

7.3.9 Creating and replacing a link

A link is a small file containing a path to another file or directory. For creating link files the filetype 'J' or 'LNK' have to be used, otherwise the referenced file will be opened. It's important to not have a CHR$(13) after the link, so watch out for the semicolon after PRINT#. When referencing a link, it's name is replaced by the links content.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,1,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,J"
OPEN ⟨*file #*⟩,⟨*device #*⟩,1,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,LNK"
OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,J,W"
OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,LNK,W"

```
10 OPEN 2,12,1,"+,J"
```

```
20 PRINT#2,"1//UTILITIES/:+64K TASM,PRG";
30 CLOSE 2
```

Listing 6: This small program creates a link in the current directory to reference '+64K TASM,PRG'. This can be handy to start the assembler with '£+' from this directory.

7.3.10 Opening a link for read

For opening link files the filetype 'J' or 'LNK' have to be used, otherwise the referenced file will be opened.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,0,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,J"
OPEN ⟨*file #*⟩,⟨*device #*⟩,0,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,LNK"
OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,J[,R]"
OPEN ⟨*file #*⟩,⟨*device #*⟩,⟨*2–14*⟩,"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*fname*⟩,LNK[,R]"

```
10 OPEN 2,12,0,"+,J"
20 GET#2,A$:IF ST AND 2 THEN 40
30 B$=B$+A$:IF ST=0 THEN 20
40 PRINT"DESTINATION:"B$:CLOSE 2
```

Listing 7: This small program reads out a link's destination

7.3.11 Opening the command channel

The command channel is identified by channel number 15.

Format:

OPEN ⟨*file #*⟩,⟨*device #*⟩,15[,"⟨*command*⟩"]

Reading the command channel returns the status message from the drive, writing to it will send commands.

7.4 CLOSE

An opened file must be closed after use. It's especially true when writing new data into a file, where data can be left in the buffer unwritten, so recent changes could be lost forever! For a newly created file this can mean the entire file, for appended or modified ones the part beyond the previous end of file, for relative files the recently written records, unless they were flushed.

Close only requires a file number.

Format:

CLOSE ⟨*file #*⟩

7.5 INPUT#, GET#

These commands are for reading from a file. For more read a BASIC manual, in assembly the calls CHKIN, CHRIN, GETIN, READ and CLRCHN can be used instead.

Format:

INPUT#⟨*file #*⟩,⟨*variables*⟩
GET#⟨*file #*⟩,⟨*variables*⟩

7.6 PRINT#

This command is for writing to a file. For more read a BASIC manual, in assembly the calls CHKOUT, CHROUT, WRITE and CLRCHN can be used instead.

Format:

PRINT#⟨*file #*⟩[,⟨*variables, strings*⟩]

7.7 CMD

This command is for redirecting output to a file. For more read a BASIC manual, in assembly the CHKOUT call can be used instead.

Format:

CMD ⟨*file #*⟩[,⟨*variables, strings* ⟩]

7.8 File operations

File operations like seek, rename, etc. are done with command channel commands. Please read section “15 Command channel”!

8 Direct access

By using direct access commands you can read and write any sector of disk. Common use of direct access commands is to access unknown filesystems, manage the partition table and to create the filesystem. It's also important in disk editor applications.

To use direct access, you need to open 2 channels, one for commands, and one for data. The command channel can be opened with the usual OPEN ⟨*lfn*⟩,⟨*device*⟩,15. The data channel is opened similar to opening normal files, except that the file name must be a hash sign. (OPEN ⟨*lfn*⟩,⟨*device*⟩,⟨*channel*⟩,"#") The channel number must be greater than 1. This "file" will be a circular data buffer holding 512 or 2048 bytes depending on the drive used. The end of buffer can be found by checking the BASIC variable ST.

In the format descriptions everything between '[' and ']' is optional, and '⟨*name*⟩' means a parameter.

8.1 Block-read

The block-read command reads the specified sector into the buffer and sets the buffer pointer to the start of buffer. It can also be used to get the ATA(PI) device identity information to find out the geometry, model number, etc. as described in ATA(PI) standards.

8.1.1 Reading from a CHS-ATA drive

Cylinder Head Sector addressing is used for older hard disks.

Format:

"B=R"+CHR$(⟨*channel #*⟩)+CHR$(⟨*head #*⟩)+
CHR$(⟨*cylinder bits 8–15 #*⟩)+CHR$(⟨*cylinder bits 0–7#*⟩)+
CHR$(⟨*sector #*⟩)

8.1.2 Reading from a LBA-ATA or ATAPI drive

Logical Block Addressing can be used with almost all devices, except old hard disks.

Format:

"B=R"+CHR$(⟨*channel #*⟩)+CHR$(⟨64+*LBA bits 24–27 #*⟩)+
CHR$(⟨*LBA bits 16–23 #*⟩)+CHR$(⟨*LBA bits 8–15 #*⟩)+
CHR$(⟨*LBA bits 0–7 #*⟩)

8.1.3 Getting device identity

Format:

"B=R"+CHR$(⟨*channel #*⟩)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)

8.2 Block-write

The block-write command writes the buffer content to the specified sector and sets the buffer pointer to the start of buffer.

8.2.1 Writing to a CHS-ATA drive

Format:

"B=W"+CHR$(⟨*channel #*⟩)+CHR$(⟨*head #*⟩)+
CHR$(⟨*cylinder bits 8–15 #*⟩)+CHR$(⟨*cylinder bits 0–7 #*⟩)+
CHR$(⟨*sector #*⟩)

8.2.2 Writing to a LBA-ATA or ATAPI drive

Format:

"B=W"+CHR$(⟨*channel #*⟩)+CHR$(⟨64+*LBA bits 24–27 #*⟩)+
CHR$(⟨*LBA bits 16–23 #*⟩)+CHR$(⟨*LBA bits 8–15 #*⟩)+
CHR$(⟨*LBA bits 0–7 #*⟩)

8.3 Buffer-pointer

The buffer pointer selects individual bytes in the buffer. Reading and writing will start on the buffer position and the buffer pointer will be incremented by the number of bytes read and written.

Format:

"B=P"+CHR$(⟨*channel #*⟩)+CHR$(⟨*position bits 0–7 #*⟩)+
CHR$(⟨*position bits 8–15 #*⟩)
"B-P:"; ⟨*channel #*⟩; ⟨*position bits 0–7 #*⟩; ⟨*position bits 8–15 #*⟩

```
10 LB=0:REM LB=64 FOR ATAPI OR LBA DEVICE!
20 OPEN 15,12,15:OPEN 4,12,4,"#"
30 AD$=CHR$(LB)CHR$(0)CHR$(0)CHR$(1)
40 PRINT#15,"B=R"CHR$(4)AD$
50 PRINT#15,"B=P"CHR$(4)CHR$(8)CHR$(0)
60 FOR A=0 TO 15:GET#4,A$:PRINTA$;:NEXT
70 CLOSE 4:CLOSE 15
```

Listing 8: This example reads in the boot sector and prints out the CFS identification string from the first sector of disk.

```
10 OPEN 15,12,15:OPEN 4,12,4,"#"
20 AD$=CHR$(0)CHR$(0)CHR$(0)CHR$(0)
30 PRINT#15,"B=R"CHR$(4)AD$
40 PRINT#15,"B=P"CHR$(4)CHR$(46)CHR$(0)
```

```
50 PRINT"FIRMWARE REVISION:":B=4:GOSUB 90
60 PRINT"MODEL NUMBER:":B=20:GOSUB 90
70 PRINT#15,"B=P"CHR$(4)CHR$(20)CHR$(0)
80 PRINT"SERIAL NUMBER:":B=10:GOSUB 90
90 CLOSE 4:CLOSE 15:END
100 FOR A=1 TO B:GET#4,A$,B$:PRINT B$A$;:NEXT
110 PRINT:RETURN
```

Listing 9: This example prints some information about the drive using the device identity.

8.4 TOC-read

The Table Of Contents holds additional information about the disc. The bits of the format parameter are described in Table 3.

The various formats of TOC are not described here, they can be found in the SCSI-MMC standard.

Format:

"B=T"+CHR$(⟨*channel #*⟩)+CHR$(⟨*format #*⟩)+CHR$(⟨*starting track #*⟩)

8.5 Sub-channel-read

This command can be used to read the current state of audio playing, the current position, media catalog number, and the ISRC. The bits of the mode and format parameters are described in Table 4 and Table 5.

For more information read the SCSI-MMC standard.

Format:

"B=S"+CHR$(⟨*channel #*⟩)+CHR$(⟨*format #*⟩)+CHR$(⟨*starting track #*⟩)+ CHR$(⟨*mode #*⟩)

Bit	Value	Meaning
6–7	0	Formatted TOC
	1	Multi-session info
	2	Raw
	3	Reserved
5		Reserved
2–4	0	Bits 6–7 select format
	1	Multi-session info
	2	Raw
	3	PMA
	4	ATIP
	5	CD-TEXT
	6–7	Reserved
1	0	LBA
	1	MSF
0		Unused

Table 3: Bits of TOC format

Bit	Value	Meaning
7		Unused
6	0	Header only
	1	Sub channel information
2–5		Unused
1	0	LBA
	1	MSF
0		Unused

Table 4: Bits of sub-channel read format

Value	Meaning
0	Reserved
1	CD current position
2	Media Catalog number (UPC/bar code)
3	International Standard Recording Code
4–255	Reserved

Table 5: Sub-channel read modes

Byte	Meaning
0	Reserved
1	Audio status, see Table 7
2	Sub-channel data length high
3	Sub-channel data length low

Table 6: Sub-channel read data header

Value	Meaning
$00	Audio status byte not supported or not valid
$11	Play operation in progress
$12	Play operation paused
$13	Play operation successfully completed
$14	Play operation stopped due to error
$15	No current audio status to return

Table 7: Sub-channel read audio status codes

Byte	Meaning
0–3	Sub-channel data header, see Table 6
4	Sub-channel data format code ($01)
5	ADR and control, see Table 9
6	Track number
7	Index in current track
8–11	Absolute CD address in LBA or MSF
12–15	Track relative CD address in LBA or MSF

Table 8: Sub-channel CD current position data format

Bit	Value	Meaning
4–7		ADR, equals to $1
3	0	Two-channel audio
	1	Four-channel
2	0	Audio track
	1	Data track
1	0	Digital copy prohibited
	1	Digital copy permitted
0	0	Audio without pre-emphasis
	1	Audio with pre-emphasis

Table 9: Sub-channel control field of CD current position

9 The File Manager

The File Manager is a software for copying, deleting, renaming, making directories, and starting programs. Working with this program is very easy. It can be started from BASIC by typing MAN. Note that it will overwrite the current program in memory.

The screen is divided into two parts called "panel"s, each containing a directory listing, with the maximal number of displayable files of 510/panel. The black line is the file selector, which can be moved by the CRSR and Fx keys. Selecting the active panel is done by the CONTROL key.

The right side of each panel shows the sum of selected files block count, the number of selected files, the used blocks in directory (or the number of free blocks on disk) and the device number with the current partition.

The current directory path is displayed on the top of screen, while the drive type and disk label is on top of each panel. The directories and files are displayed with different color for easier recognition.

To change the current drive of a panel press the C= key together with a number key, and it will select drives 10, 11, ..., 17, 8, and 9. If the same drive is selected as in the inactive panel the working directory will became the same as well, otherwise the root directory is entered. Directory reload is done with the 1 key in case of disk change.

Tagging files for operations with multiple files is done with the DEL key, the selected files will be highligted. The + key selects and the − deselects all matching filenames for the given pattern. The key ∗ inverts the current selection. These three keys can be combined with SHIFT to include directories too.

Copying files can be done with key 5, while deleting is done with 8. Both operations can work on multiple files and on subdirectories. If no files are selected then copying and deleting will operate on the file where the selector currently is.

File rename is done with the 6 key, this can also change the filetype on IDE64 drives. Directory creation is done with the key 7.

When using 6 with tagged files it will not rename the files, but move them recursively. This is accomplished either with copy and delete, or by using the IDEDOS fast move if it's available and both the source and destination reside on the same drive and partition.

To send commands to ordinary disk drives (e.g. for formatting or validating a disk) press 9. This is also useful on virtual filesystems.

To jump quickly on a file press it's first letter. (alphabetic characters only)

There's also a shortcut to eject the disc from a CD-ROM, DVD, LS-120 or Zip drive by pressing the ↑ key. Reloading of the disc is automatically done when reloading the directory. To exit to BASIC press ←.

Entering directories and loading files are done with the RETURN key. If a file is associated with a plugin then the plugin will be started instead. Going into the parent directory is done by pressing RETURN on the directory entry '..'. Or you can also use the '.' key for entering the parent directory and key '/' to enter the root directory.

Partition selection is similar like entering the parent directory, but you have to be in the root directory. (press key '/' and then '.' to quickly reach it) The partition selection directory can be identified by the partition number 255 displayed on the side. (and of course there's no '..' entry)

The manager remembers the last used directories, partitions and

device numbers across invocations unless the computer is turned off.

If you do not like the default colors then it's possible to change it in the setup utility. If you do not like the lowercase character set (e.g. want to look at a nice directory art), use the well known C= + SHIFT combo.

Files are unsorted for 1541, 1570, 1571, 1581 while any other drive will have it's directory displayed alphabetically, and directories will be sorted to the top. Do not put more than 510 files in a directory or the rest will not be displayed.

Key	Function
LEFT, RIGHT	Cursor move
HOME or F1 or CONTROL + A	Cursor to beginning
F7 or CONTROL + E	Cursor to end
INS	Insert space
DEL	Delete before cursor
CONTROL + X	Delete at cursor
CLR or CONTROL + L	Delete all
CONTROL + K	Delete to end
CONTROL + U	Delete to beginning
C= + INS	Toggle insert/overwrite
STOP	Abort editing
RETURN	Finish editing

Table 10: Manager input window keys

For question windows with multiple selections (e.g. YES/NO/ALL) press the first letter of your choice. Simple message windows will accept any key. If you hold down the key long enough, you can see the window below the message. (e.g. disk error while copying, but

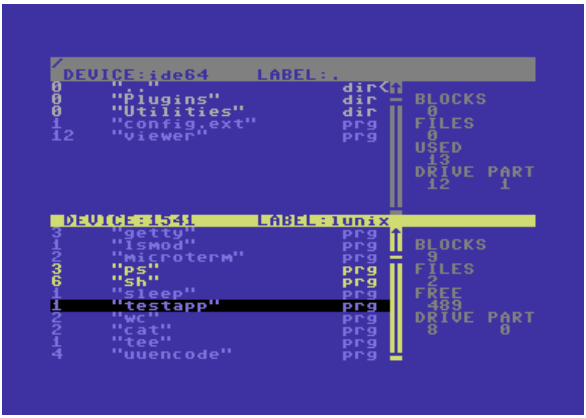


Figure 6: The File Manager

which file had this error?)

The recursive copy operation is not the “ultimate backup tool”, as it can’t copy relative files, preserve creation or modification date or file attributes.

Fast copying with a serial bus drive is only possible if it’s supported by the built in floppy speeder (if enabled) or it’s capable of the JiffyDOS or Dolphin DOS protocol. Copying between serial drives will use the normal routines.

9.1 Plugins

External programs called by the file manager can extend it’s functionality in a number of ways. Plugins can show text and various graphic formats, play SID files and animations, extract archives and

disk images, fire up an assembler with the file, or just anything one can imagine!

These external programs are started by pressing RETURN, 2, 3 or 4 on a file. The action taken when these keys are pressed is determined by the file called '1//:MAN,USR'. The manager looks for these files on the system drive which can be set in the setup utility.

9.1.1 Plugin interface

Plugins are machine code files compiled to $1000. A sample source code can be seen in Listing 10.

The interface is defined as follows:

$1000 When started here it should display some kind of prompt for getting the filename. Not used by the manager.

$1003 The manager starts the plugins at $1003 with the accumulator loaded with the length of filename, the X register with the lower, the Y with the higher byte of the address to the filename. $BA contains the current device number the file is on. The filename always includes the file type separated by a comma on the end.

```
        *= $1000

        jmp printc
        jmp start

txt     .text 13,"filename:",0
```

```
printc  lda #<txt
        ldy #>txt
        jsr $ab1e       ;print prompt
        ldy #255
oqu     iny
        jsr $ffcf       ;get input
        sta $200,y
        cmp #13
        bne oqu
        tya             ;setup fake manager
        ldx #<$200      ;filename parameter
        ldy #>$200

start   jsr $ffbd       ;setup filename
        lda #filenumber ;file number (1-255)
        ldx $ba         ;actual device number
        ldy #secaddy    ;secondary address
        jsr $ffba
        jsr $ffc0       ;open file
        ...             ;play movie
        lda #filenumber ;file number (1-255)
        jsr $ffc3       ;close file
        rts             ;done
```

Listing 10: Plugin sample source

The plugin must not destroy $0002–$0333 badly and must not modify $0800–$0FFF! Please leave the VIC II, SID, CIA, etc. after exit of the plugin in a usable state...

9.1.2 Virtual filesystem interface

Virtual filesystem drivers are used to display, execute and modify content of container files like D64, T64, etc. The content of files is displayed just like a normal directory in the manager, and all the normal operations like executing, copying, deleting, renaming, or custom commands are possible.

A virtual filesystem driver's loading address must be `$D000` and it's entry points are defined as follows:

`$D000` Initialization, this is called when entering an image file.

This routine usually opens the image file and sets up a jump table for calling kernal routines in the cassette buffer.

The filename, filename length, logical file number, device number and secondary address are already set up in advance, so that opening the image file can be done without any further work.

For the command channel the next higher logical file number can be taken.

`$D003` Exit, this is called when leaving an image file.

This should close all files opened for image handling at initialization.

`$D006` Directory list, this is called for listing the content.

The X and Y registers contain the start address in low and high format, while the accumulator holds the maximal number of pages (256 bytes) the directory list can occupy.

The memory content produced should be the same as when loading a BASIC directory list, including header and blocks free lines.

On return the X and Y registers contain the address after the last byte.

If the carry is set then the accumulator contains the kernal error number, as defined in Table 21.

If the carry is clear the accumulator controls the sorting of the list in the panel. Zero means sorted, anything else unsorted.

The status at $90 must be set to $40 on success.

$D009 Command channel command, used to operate on files.

The filename and filename length is already set up to point to the command. The commands are normal DOS commands and should be parsed like that.

The X and Y registers contain the address where the DOS error message should be written. The message starts with a two digit leading error code, is terminated by `CHR$(13)`, and is not longer than 64 bytes.

It's recommended to implement "UI" for returning plugin version information.

$D00C Open file in preparation for read/write.

The filename and filename length is set up in advance. If the secondary address is zero it's for reading an existing file, if it's one, then it's for creating a file.

The filename can contain a leading partition number or rewrite marker (@).

If the carry is set then the accumulator contains the kernal error number, as defined in Table 21.

The file exists, file not found, ok, write protect on, etc. messages should be reported through the command channel interface ($D009).

$D00F Load and execute file.

The filename and filename length is set up in advance. If the secondary address is zero the file is loaded to the BASIC start ($2B), if it's one, then to the address specified by the first two bytes of the file.

There is no return from this function, so the procedure must be done like on exit ($D003).

The file should be automatically executed by RUN if it's loaded with secondary address zero.

$D012 Read from file used while copying, the file is opened before.

The X and Y registers contain the length of read in low and high format, the accumulator holds the address on the zero page word which contains the starting address of buffer.

The result is to fill the buffer with the file content. On return the X and Y registers contain the number of bytes read in low and high format.

If there was an error the carry is set, the reason is put to the command channel.

The status at $90 must be set to $40 on end of file.

$D015 Write to file used while copying, the file is opened before.

The X and Y registers contain the length of read in low and high format, the accumulator holds the address on the zero page word which contains the starting address of buffer.

The result is to write the buffer content to the file. On return the X and Y registers contain the number of bytes written in low and high format.

If there was an error the carry is set, the reason is put to the command channel.

$D018 Close the file opened for copying

This routine is called at the end of copying for flushing file buffers.

The virtual filesystem driver is free to use the cassette buffer at $334–$3FF and the upper memory area $D000–$FFFF. If zero page addresses are used, they must be backed up, otherwise the manager could malfunction.

9.2 Manager configuration file

The configuration file can be found in the root directory of the system drive and it's called 'MAN' with filetype 'USR'. The manager only loads it on the first invocation, so if you want that your changes take affect immediately, then start the manager as 'MAN!'.

This file controls the plugin assignment based on powerful wildcard matching. Each line ends on CHR$(0). The first line contains

the plugin directory, it's after a fake 2 bytes loading address. The path must contain the partition number and the full path from the root directory ending with ':'. Then pairs of lines follow, where the first is the plugins name, and the second the wildcard pattern. An empty line terminates the file. The special code '=' alone is a key definition separator.

For the easy creation of the configuration file a BASIC program is included here in Listing 11.

```
0 DR=PEEK(186):REM GET LAST USED DRIVE NUMBER
10 OPEN 1,DR,1,"@1//:MAN,USR":PRINT#1,"."
20 N$=CHR$(0):REM NULL SEPARATOR
30 PRINT#1,"1//PLUGINS/:"N$;:REM PLUGIN DIRECTORY
40 REM RETURN KEY SECTION BEGINS
50 PRINT#1,"SID"N$"*,SID"N$;:REM SID PLAYER
55 K$=CHR$(129)
60 PRINT#1,"KLA"N$K$"PIC ? *,PRG"N$;:REM KOALA
70 PRINT#1,"DRL"N$"*.DRL"N$;:REM DRAZLACE
80 PRINT#1,"D64"N$"*,D64"N$;:REM D64 WRITER
90 REM  ********************
100 REM *** MORE PRUGINS ***
110 REM ********************
500 PRINT#1,"="N$;:REM KEY 2 SECTION BEGINS
510 REM ********************
520 REM *** MORE PRUGINS ***
530 REM ********************
590 PRINT#1,"MENU"N$"*" N$;:REM MENU
600 PRINT#1,"="N$;:REM KEY 3 SECTION BEGINS
610 PRINT#1,"D64L"N$"*,D64" N$;:REM D64 LISTER
620 REM ********************
630 REM *** MORE PRUGINS ***
640 REM ********************
690 PRINT#1,"VIEWER"N$"*"N$;:REM VIEWER
700 PRINT#1,"="N$;:REM KEY 4 SECTION BEGINS
710 REM ********************
720 REM *** MORE PRUGINS ***
```

```
730 REM ********************
790 PRINT#1,"VI65"N$"*"N$;:REM EDITOR
800 PRINT#1,N$;:CLOSE 1:REM CLOSE FILE
```

Listing 11: MAN,USR generator source

The program first opens the configuration file, and removes the old one if there was any. Also it writes the fake loading address. Line 30 defines '/PLUGINS/' as the plugin directory on partition 1. Line 50 is an example of a filetype matching. All files with filetype 'SID' will be played by the '1//PLUGINS/:SID' plugin, when RETURN is pressed. Line 60 associates all 'PRG' type files beginning with 'APIC' and a letter between spaces to the Koala painter viewer plugin. Line 70 associates all files ending on '.DRL' to the Drazlace viewer. Line 80 associates all files with filetype 'D64' to the d64 writer. Line 500 starts definitions for key 2. Line 590 associates all files to the menu program, when key 2 is pressed. Line 600 starts definitions for key 3. Line 610 associates all files with filetype 'D64' to the d64 lister. Line 690 associates all files to the viewer program, when key 3 is pressed. Line 700 starts definitions for key 4. Line 790 associates all files to the editor program, when key 4 is pressed. In line 800 the end of definitions marker is written, and then the file is closed.

The pattern matching is case sensitive, so it's not a bad idea to add both lowercase and uppercase variants to avoid problems with badly written CDs.

Save the configuration creator program, in case you want to modify or add some more plugins to you manager configuration file sometime later.

Key	Function
←	Exit
CONTROL	Change panel
UP , DOWN	Move cursor
F1 or LEFT	Page Up
F7 or RIGHT	Page Down
1	Refresh dir
2 , 3 , 4	Execute plugins
5	Copy file(s)
6	Rename or move file or directory
7	Create directory
8	Delete file(s)
9	Send DOS command
.	Go into parent dir
/	Go into root dir
DEL	Toggle selection of file
F2 or HOME	Go on top of listing
F8 or CLR	Go to end of listing
RETURN	Enter directory or LOAD"FILE",X and RUN or start viewer plugin
SHIFT + RETURN	LOAD"FILE",X,1
↑	Eject medium
*	Invert file selection (SHIFT to include dirs)
+	Select files (SHIFT to include dirs)
−	Deselect files (SHIFT to include dirs)
C= + 0–9	Select drive
C= + SHIFT	Select character set
A–Z	Quick jump to filename

Table 11: Manager keys

10 Using the monitor

The monitor is designed to be fast and simple, while also powerful for debugging and fixing. It's possible to edit every single byte in memory and I/O space without conflicting with internal variables of the monitor itself.

Illegal opcodes are supported in the C64 version, 65816 opcodes (emulation mode only) in the SuperCPU version.

The default radix for all numbers and addresses is hexadecimal (the '$' prefix is optional). For decimal numbers use the '+' prefix. Octals are prefixed by '&' and binary numbers with '%'. This feature is not limited to addresses only, but can also be used in the parameters of machine code instructions.

It's possible to give the location of a pointer to an address by using the indirection prefix '∗'. For example 'M ∗2B' will display the start of basic program memory by using the pointer at $2B to find out the real address.

10.1 Starting the monitor

The monitor is started by hitting C= + RESTORE at the same time, or by executing a BRK instruction, or pressing LEFTSHIFT + RESET, or alternatively by using the command SYS 0. If C128 keyboard is enabled it's possible to use HELP to start the monitor.

To recover from died monitor (not very likely) press LEFTSHIFT + RESET. Memory won't get distorted.

To rip from games, demos, etc. press LEFTSHIFT + RESET while the program is running. (not a real freezer, but at least it's possible to rip IFLI pictures...)

10.1.1 Ripping on C64

A few bytes from stack (`$1F3`–`$1FF`), the contents of both CIAs, the memory configuration at `$00`, `$01` and the processor registers (`A`, `X`, `Y`, `SR`, `SP`) will be lost, otherwise the memory is not touched. (`ADDR` will not have a valid restart address, instead it contains `$FCE2`)

10.1.2 Ripping on SuperCPU

A few bytes from stack (`$1F3`–`$1FF`), the contents of VIC II registers `$D020`–`$D02E`, and zero page addresses `$99`, `$9A` will be lost, otherwise the memory is not touched. Due to end of stack corruption you may have problems with the restarted program. (`ADDR` will have a valid restart address!) Use SuperCPU reset button, stopping the program for a short time before freeze may be necessary.

To use the monitor like a real freezer monitor you must make sure that the NMI vector (`$318`) won't get overwritten in RAM. If it's OK, press C= + RESTORE to freeze the program. The chip states, processor registers and memory are preserved.

In syntax descriptions everything between '[' and ']' is an optional parameter, while '⟨' and '⟩' means a required parameter.

10.2 Disk commands

10.2.1 Select work drive

Command: o, @#
Purpose: Selects working drive for disk operations.
Syntax: o ⟨*drive*⟩, @#⟨*drive*⟩

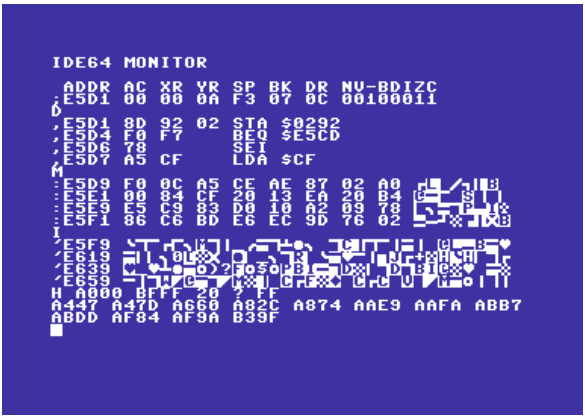

IDE64 MONITOR
ADDR AC XR YR SP BK DR NV-BDIZC
.E5D1 00 00 0A F3 07 0C 00100011
D
,E5D1 8D 92 02 STA $0292
,E5D4 F0 F7 BEQ $E5CD
,E5D6 78 SEI
,E5D7 A5 CF LDA $CF
M
:E5D9 F0 0C A5 CE AE 87 02 A0
:E5E1 00 84 CF 20 13 EA 20 B4
:E5E9 E5 C9 83 D0 10 A2 09 78
:E5F1 86 C6 BD E6 EC 9D 76 02
I
;E5F9
;E619
;E639
;E659
H A000 BFFF "0 ? FF
A447 A47D A660 A82C A874 AAE9 AAFA ABB7
ABDD AF84 AF9A B39F

Figure 7: The IDE64 builtin monitor

Device numbers for the 'O' command must be entered as hexadecimal unless a radix prefix is used.

The '@#' command always requires a decimal number, no radix prefix is allowed.

The device number is used in all disk access commands. The default is the system drive's device number selected in the setup.

Example:

Select device 12.

```
O C
```

Select device 13:

```
@#13
```

10.2.2 DOS command

Command: @
Purpose: Read error channel and send DOS commands.
Syntax: @⟨*command*⟩

Examples:

Print error channel:

```
@
00, OK,000,000,000,000
```

Send a DOS command:

```
@CD:SOURCE
00, OK,000,000,000,000
```

10.2.3 Directory

Command: @$
Purpose: Display directory of current working drive.
Syntax: @$[⟨*pattern*⟩]

Example:

Display directory:

```
@$T*
2 "TEST            " IDE64
13105"TRANCEANDACID"
5    "TOD"           ASM
13110 BLOCKS USED.
```

10.2.4 Save

Command: S and SB
Purpose: Save program to disk and save a memory area to disk.
Syntax: S"NAME" ⟨*start address*⟩ ⟨*end*⟩ [⟨*start2*⟩]
SB"NAME" ⟨*start address*⟩ ⟨*end*⟩

Examples:

Save main file to disk, save sprites as 512 byte raw file without start address, and save 1024 bytes of data from $3000 with start address $2200.

Note that the first parameter is not the device number!

```
S"GAME" 0801 21FF
OK

SB"SPRITES" 2000 21FF
OK

S"TO2000" 3000 33FF 2200
OK
```


10.2.5 Load

Command: L and LB
Purpose: Load a program or memory area from disk.
Syntax: L"NAME" [⟨*start address*⟩]
LB"NAME" ⟨*start address*⟩

Examples:

Load the main program from disk, and link in sprite data, which is a 512 byte raw file without start address.

Note that the first parameter is not the device number!

```
L"GAME"
0801-21FF OK

LB"SPRITES" 2200
2200-23FF OK
```

10.2.6 Verify

Command: V and VB
Purpose: Verify a program or memory area from disk.
Syntax: V"NAME" [⟨*start address*⟩]
VB"NAME" ⟨*start address*⟩

Examples:

Verify the main program from disk, and the sprite data, which is a 512 byte raw file without start address.

Note that the first parameter is not the device number!

```
V"GAME"
0801-21FF OK

VB"SPRITES" 2200
 2312 2313
2200-23FF OK
```

10.2.7 Freeze

Command: S
Purpose: Save current machine state to disk.
Syntax: S"NAME"

Freeze (saves 64 KiB RAM and I/O) memory to disk. The actual bank must be 0–7.

Example:

Freeze memory from disk.

```
S"MYFREEZE"
OK
```

10.2.8 Defreeze

Command: K
Purpose: Restore machine state from disk.
Syntax: K"NAME"

Defreeze memory from disk (loads 64 KiB RAM and I/O). The actual bank must be 0–7.

Example:

Defreeze memory from disk.

```
K"MYFREEZE"
0000-FFFF OK
```


10.3 Display and modify memory

10.3.1 Registers

Command: R
Purpose: Shows the processor registers.
Syntax: R

To modify the registers and flags, change the line beginning with ';', and press RETURN. ADDR is the address where the program continues (see X!), AC XR YR SP are the Accumulator, X register, Y register and Stack pointer, BK is the currently edited bank selector (see B!), DR is the device number used in disk access commands (see O!), NV-BDIZC are the processor flags.

Example:

Display registers.

```
R
  ADDR AC XR YR SP BK DR NV-BDIZC
;E5D1 00 00 0A F3 07 0C 00100010
```

10.3.2 I/O chip registers

Command: IO
Purpose: Shows the I/O chip registers.
Syntax: IO

This command is a shortcut to quickly display VIC II and CIA registers.

Example:

Display I/O chip registers.

```
IO
-D000 00 00 00 00 00 00 00 00
-D008 00 00 00 00 00 00 00 00
-D010 00 1B 37 00 00 00 C8 00
-D018 15 71 F0 00 00 00 00 00
-D020 FE F6 F1 F2 F3 F4 F0 F1
-D028 F2 F3 F4 F5 F6 F7 FC FF

-DC00 7F 00 FF 00 25 40 FF FF
-DC08 00 00 00 01 00 81 01 08

-DD00 C7 00 3F 00 FF FF FF FF
-DD08 00 00 00 01 00 00 08 08
```

10.3.3 Assemble

Command: A
Purpose: Enter assembly code.
Syntax: A ⟨*address*⟩ ⟨*mnemonic*⟩ [⟨*operand*⟩]

Illegal instructions are supported and emulation mode 65816 instructions too.

Examples:

Enter a few assembly instructions.

```
A1000 EE 20 D0 INC $D020
A1003 4C 00 10 JMP $1000
A1006 B3 30    LAX ($30),Y
A1008 NOP
```

These are legal instruction entering forms:

```
A1000 INC D020
A1003 LDA#$3
A1006 NOP:VSFV
```

10.3.4 Disassemble

Command: D
Purpose: Disassemble machine code into assembly.
Syntax: D [⟨*address 1*⟩ [⟨*address 2*⟩]]

Pressing RETURN on modified hex bytes or on modified disassembly changes memory. (the cursor position selects what happens) To slow down listing hold CONTROL, to stop press STOP, to pause listing press anything else. If the whole screen is filled and you want to get an empty line then go into the last line and press SHIFT + RETURN. (or clear the screen)

Examples:

Disassembly from last address: D.
Disassembly a few lines: D 1234.
Disassembly continuously forward from: D1234-.

Disassembly continuously backwards from: D-1234.
Disassembly between addresses: D1234-5678.

Simply display memory at $1000.

```
D 1000
,1000 EE 20 D0 INC $D020
,1003 4C 00 10 JMP $1000
,1006 B3 30    LAX ($30),Y
```

10.3.5 Display as hexadecimal

Command: M
Purpose: Dump memory in hex and PETSCII.
Syntax: M [⟨*address 1*⟩ [⟨*address 2*⟩]]

Press RETURN to enter the modified hex values or PETSCII text into memory. Possible parameters are the same as for D.

Example:

Display memory from $E478.

```
M E478
:E478 20 2A 2A 2A 2A 20 43 4F  **** CO
:E480 4D 4D 4F 44 4F 52 45 20 MMODORE
:E488 36 34 20 42 41 53 49 43 64 BASIC
:E490 20 56 32 20 2A 2A 2A 2A  V2 ****
```

10.3.6 Display as PETSCII

Command: I
Purpose: Display memory as PETSCII.
Syntax: I [⟨*address 1*⟩ [⟨*address 2*⟩]]

Press RETURN to enter the modified text into memory. Some PETSCII values have the same screen code, in this case the original value is not overwritten, to preserve mixed code. Possible parameters are the same as for D.

Example:

Display memory from $A0A0.

```
i a0a0
'a0a0 DfoRnexTdatAinput_inpuTdiMreaDle
'a0c0 TgotOruNiFrestorEgosuBreturNreMs
'a0e0 toPoNwaiTloaDsavEverifYdeFpokEpr
'a100 int_prinTconTlisTc1RcmDsySopeNc1
```

10.3.7 Display as screen code

Command: J
Purpose: Display memory as screen code.
Syntax: J [⟨*address 1*⟩ [⟨*address 2*⟩]]

Press RETURN to enter the modified text into memory. (I∗ is available for MK7 addicts) Possible parameters are the same as for D.

Example:

Display memory from $400.

```
J 400
.0400
.0420      **** COMMODORE 64 BA
.0440 SIC V2 ****
.0460                   64K RAM
```

10.3.8 Display as binary

Command: EC
Purpose: Display memory as binary.
Syntax: EC [⟨*address 1*⟩ [⟨*address 2*⟩]]

Press RETURN to enter the modified bytes into memory. '.' means 0, anything else (not a space) is 1. Possible parameters are the same as for D.

Example:

Display memory from the character ROM at address $D008.

```
B 3
EC D008
.D008 ...##...
.D009 ..####..
.D00A .##..##.
.D00B .########.
.D00C .##..##.
.D00D .##..##.
.D00E .##..##.
.D00F ........
```


10.3.9 Display as sprite

Command: ES
Purpose: Display memory as sprite.
Syntax: ES [⟨*address 1*⟩ [⟨*address 2*⟩]]

Press RETURN to enter the modified bytes into memory. '.' means 0, anything else (but not a space) is 1. Possible parameters are the same as for D.

Example:

Display a sprite from $A00.

```
ES A00
]0A00 ........................
]0A03 ........................
]0A06 ........................
]0A09 ........................
]0A0C ........................
]0A0F ........................
]0A12 ........................
]0A15 ........................
]0A18 ..........##########.....
]0A1B ........##############....
]0A1E ......################....
]0A21 .....##########.....#....
]0A24 ....###########.....#....
]0A27 ...###########.......#....
]0A2A ...###########..#...#....
]0A2D ...###########...#..##....
]0A30 ....###.......#..##.....
]0A33 .....#.............#.....
]0A36 .....#............##....
]0A39 ......#....#......#.....
]0A3C ........##...#....#.....
```

10.3.10 Backtrace

Command: `BT`
Purpose: Display call trace.
Syntax: `BT`

Example:

Do a backtrace. $1237, $1233 and $1230 are the caller JSR addresses, $30 is some pushed data.

```
A1230 20 33 12 JSR $1233
A1233 20 36 12 JSR $1236
A1236 08       PHP
A1237 20 3A 12 JSR $123A
A123A 00       BRK
A123B

G 1230

BRK EXCEPTION

 ADDR AC XR YR SP BK DR NV-BDIZC
;1230 10 00 0A C6 07 0D 00110000
```

```
BT
1237 30 1233 1230 7A E3 1009 050E 10
```

10.4 Execution control

10.4.1 Go

Command: G
Purpose: Execute a routine, and return to monitor.
Syntax: G ⟨*address*⟩

When program terminates it tries to return to monitor with a BRK.

Example:

Start program at $080D.

```
G 80D
```


10.4.2 Exit

Command: X
Purpose: Exit monitor and continue execution.
Syntax: X

As the IDE64 monitor is always in freeze mode, exit will continue the interrupted program. If you want to get back to the BASIC prompt, use Q.

Example:

Continue program execution at ADDR.

```
IDE64 MONITOR

 ADDR AC XR YR SP BK DR NV-BDIZC
;E5D1 00 00 01 F2 07 08 00100010
X
```

10.4.3 Quit

Command: `Q`
Purpose: Return to BASIC prompt from monitor.
Syntax: `Q`

Example:

Try to exit to BASIC prompt. Useful after hitting a `BRK`.

```
BRK EXCEPTION

 ADDR AC XR YR SP BK DR NV-BDIZC
;1230 10 00 0A C6 07 0D 00110000
Q
```

10.5 Memory area commands

10.5.1 Transfer

Command: `T`
Purpose: Copy a memory area to another address.
Syntax: `T` ⟨*start address*⟩ ⟨*end*⟩ ⟨*destination*⟩

Overlapping areas are supported.

Example:

Copy memory from `$400`–`$7FF` to `$C00`–`$FFF`.

```
T 400 7FF C00
```

10.5.2 Compare

Command: `C`
Purpose: Compare a memory area with another.
Syntax: `C` ⟨*start address*⟩ ⟨*end*⟩ ⟨*start2*⟩

Example:

Compare memory from $2000–$3FFF with $E000–$FFFF.

```
C 2000 3FFF E000
 3FC0 3FC1 3FC2
```

10.5.3 Fill

Command: F
Purpose: Fill a memory area with the specified pattern.
Syntax: F ⟨*start address*⟩ ⟨*end*⟩ ⟨*pattern*⟩

Example:

Fill memory $2–$FFFF with $00.

```
B 4
F 2 FFFF 0
```

Fill memory with PETSCII text.

```
F 1000 1FFF "TEST"
```

Fill memory with screen code text.

```
F 400 7E7 'TEST'
```

10.5.4 Hunt

Command: H
Purpose: Search memory for a specific pattern.
Syntax: H ⟨*start address*⟩ ⟨*end*⟩ ⟨*pattern*⟩

'?' is a one character wildcard matching everything.

Example:

Search memory for instructions accessing $277.

```
H E000 FFFF ? 77 2
 E5B4 E5BC EB3C
D E5B4
,E5B4 AC 77 02 LDY $0277
,E5B7 A2 00    LDX #$00
,E5B9 BD 78 02 LDA $0278,X
```

Search memory for PETSCII text.

```
H E000 FFFF "COMM"
 E47E
```

Search memory for screen code text.

```
H 400 800 'COMM'
 0431
```


10.6 Miscellaneous

10.6.1 Bank

Command: B
Purpose: Select memory configuration and area.
Syntax: B ⟨*bank*⟩

Banks 0–7 have the same meaning like the first 3 bits of $01, banks 8–1E selects the drive's memory with that device number.

Example:

Select bank 4.

```
B 4
```

10.6.2 RAM/ROM

Command: ∗
Purpose: Change between banks 4 and 7.
Syntax: ∗

Example:

Quick switch between RAM and ROM (bank 4 and 7)

```
  ADDR AC XR YR SP BK DR NV-BDIZC
;EAB4 03 03 15 EA 07 08 00100101
*
RAM
R
  ADDR AC XR YR SP BK DR NV-BDIZC
;EAB4 03 03 15 EA 04 08 00100101
```


10.6.3 Number convert

Command: N
Purpose: Calculate an expression and display the result.
Syntax: N ⟨*expression*⟩

The result is displayed in decimal, octal, binary, screen code and PETSCII.

Example:

Calculate something.

```
N $0E00+23*SIN(1)
3603 0E13 0000111000010011 SN SN
```

10.6.4 View

Command: W
Purpose: Look at screen.
Syntax: W

Example:

Useful for finding the video bank after LEFTSHIFT + RESET.

```
W
```

10.6.5 Restore I/O vectors

Command: IV
Purpose: Restore I/O vertors.
Syntax: IV

Example:

Restore vectors at $314–$333.

```
IV
OK
```

10.6.6 Address stack

Command: ←
Purpose: Push addresses to the “address stack”.
Syntax: ← ⟨*addresses*⟩

Search for something, then put a ‘←’ before the address list, press RETURN (push), then list with ‘D↑’, or with something else (pop). The address stack is 8 address deep only.

Example:

An example using the astack.

```
H E000 FFFF 20 BA FF
←E1DD E1F0 E22B E23F E24E
D↑
,E24E 20 BA FF JSR $FFBA
,E251 20 06 E2 JSR $E206
,E254 20 0E E2 JSR $E20E
D↑
,E23F 20 BA FF JSR $FFBA
,E242 20 06 E2 JSR $E206
,E245 20 00 E2 JSR $E200
D↑
,E22B 20 BA FF JSR $FFBA
,E22E 20 06 E2 JSR $E206
,E231 20 00 E2 JSR $E200
D↑
,E1F0 20 BA FF JSR $FFBA
,E1F3 20 06 E2 JSR $E206
,E1F6 20 00 E2 JSR $E200
D↑
,E1DD 20 BA FF JSR $FFBA
,E1E0 20 06 E2 JSR $E206
,E1E3 20 57 E2 JSR $E257
```


10.6.7 Scrolling

Pressing F3 and F5 displays previous and next lines of current display mode (e.g. disassembly, screencode, etc.) The F1 and F7 keys do the same, but scroll a half screen on one press.

Real crackers use F2 and F8 for fast code search, and pause with any key or stop with STOP, as soon as something interesting turns up. ;-)

Example:

Type `D FCE2` then hold F3 till the code before $FCE2 appears. (other keys work similar)

```
,FCDD D0 02     BNE $FCE1
,FCDF E6 AD     INC $AD
,FCE1 60        RTS
```

```
,FCE2 A2 FF     LDX #$FF
,FCE4 78        SEI
,FCE5 9A        TXS
,FCE6 D8        CLD
,FCE7 20 02 FD JSR $FD02
```

10.6.8 Using freeze points

There's support for two freeze points and a zeropoint. Unlike other monitor programs the freeze and zero point instructions are restored even if they were relocated since, but only the triggered one. In the C64 version it's possible to call the monitor with freeze points even from configurations like 5. (I/O area must be available)

In the next example a freeze point is set at $1000. The SF command will insert a JSR call to the current location. The original code will be restored[9] when control returns to monitor by the JSR freeze point. The I/O area must be available to return.


```
D 1000
,1000 SF 20 D0 INC $D020
,1003 4C 00 10 JMP $1000
,1006 B3 30    LAX ($30),Y
```

SZ will set a zero point (BRK) at $1000. Works similar to the freeze point, but only one byte long. Of course the I/O area must be available to return, and the BRK vector must be correct.

```
D 1000
,1000 SZ 20 D0 INC $D020
,1003 4C 00 10 JMP $1000
,1006 B3 30    LAX ($30),Y
```

A freeze point or zero point can be restored manually by using the R command. After pressing RETURN the original code appears, and F5 can be used to update the following lines on screen.

[9] Only that one will be restored which gets executed.

```
D 1000
,1000 R0 F7 DE JSR $DEF7
,1003 4C 00 10 JMP $1000
,1006 B3 30    LAX ($30),Y
```

Key	Function
F1	Scroll half screen up
F2	Scroll continuously up
F3	Scroll up
F5	Scroll down
F7	Scroll half screen down
F8	Scroll continuously down
CRSR	Cursor move
INS	Insert space
DEL	Delete before cursor
HOME	Cursor to top left
CLR	Clear screen
RVSON, RVSOFF	Set or reset inverse mode
RETURN	Execute line
SHIFT + RETURN	Go to beginning of next line
STOP	Stop operation
CONTROL + A	Cursor to beginning
CONTROL + E	Cursor to end
CONTROL + I	Cursor to next tab
CONTROL + J	Cursor to next line
CONTROL + K	Delete to end
CONTROL + L	Delete line
CONTROL + U	Delete to beginning
CONTROL + X	Delete at cursor

C= + SHIFT	Select character set
C= + INS	Toggle insert/overwrite
C= + RETURN	Skip lines starting on left

Table 12: Monitor editor keys

Command	Function
@ [⟨*command*⟩]	Disk command, status
@$ [⟨*pattern*⟩]	Directory list
@# ⟨*decimal number*⟩	Select work drive
A ⟨*address*⟩ ⟨*mnemonic*⟩	Assemble to machine code
B ⟨*bank*⟩	Select memory bank
*	Select RAM/ROM bank
BT	Backtrace
C ⟨*start*⟩ ⟨*end*⟩ ⟨*start2*⟩	Compare memory
D [⟨*start*⟩ [⟨*end*⟩]]	Disassemble
, ⟨*start*⟩ ⟨*data*⟩ ...	Enter data and disassemble
EC [⟨*start*⟩ [⟨*end*⟩]]	Dump as character
[⟨*start*⟩ ⟨*binary*⟩	Enter character data
ES [⟨*start*⟩ [⟨*end*⟩]]	Dump as sprite
] ⟨*start*⟩ ⟨*binary*⟩	Enter sprite data
F ⟨*start*⟩ ⟨*end*⟩ ⟨*pattern*⟩	Fill with pattern
G ⟨*address*⟩	Execute at address
H ⟨*start*⟩ ⟨*end*⟩ ⟨*pattern*⟩	Search data/any/text
I [⟨*start*⟩ [⟨*end*⟩]]	Dump in PETSCII
´ ⟨*start*⟩ ⟨*text*⟩	Write PETSCII data
IO	Dump I/O registers
– ⟨*start*⟩ ⟨*hex*⟩ ...	Write data to I/O area

IV	Restore I/O vectors
J [⟨*address 1*⟩ [⟨*address 2*⟩]]	Dump in screen code
. ⟨*start*⟩ ⟨*text*⟩	Enter screen code
K "NAME"	Defreeze memory
L "NAME" [⟨*start*⟩]	Load program
LB "NAME" ⟨*start*⟩	Load binary data
M [⟨*start*⟩ [⟨*end*⟩]]	Dump in hex and PETSCII
: ⟨*start*⟩ ⟨*hex*⟩ . . .	Enter hex or PETSCII data
N ⟨*expression*⟩	Conversion and calculator
O ⟨*number*⟩	Select work drive
R	Show registers
; ⟨*pc*⟩ ⟨*ac*⟩ ⟨*xr*⟩ . . .	Change registers
S "NAME"	Freeze memory
S "NAME" ⟨*start*⟩ ⟨*end*⟩ [⟨*start2*⟩]	Save program
SB "NAME" ⟨*start*⟩ ⟨*end*⟩	Save binary data
T ⟨*start*⟩ ⟨*end*⟩ ⟨*destination*⟩	Copy memory
V "NAME" [⟨*start*⟩]	Verify program
VB "NAME" ⟨*start*⟩	Verify binary data
X	Continue program
Q	Exit to BASIC warm start
← ⟨*address*⟩ ⟨*address*⟩	Push address(es) to astack
↑	Pop address from astack

Table 13: Monitor commands

11 DOS Wedge

These are one or two character long commands to speed up some frequently typed command at the BASIC prompt. Enable or disable this feature in the setup utility under “3.1.7 Use DOS wedge”.

11.1 @ – DOS command

It sends the command string (if any) to the last used device and then prints the error channel message.

Examples:

Print error channel:

```
@
00, OK,000,000,000,000

READY.
```

Delete a file:

```
@S:MYFILE
01, FILES SCRATCHED,001,000,000,000

READY.
```

11.2 @# – select device

Selects the specified device as last used device.

Example:

Select device 8:

```
@#8

READY.
```

11.3 @$ – list directory

Display directory without affecting memory

Example:

List ‘PRG’ files beginning with ‘S’ ending with ‘.DRL’.

```
@$S*.DRL=P

1 "TEST                " 10,64
33   "SHUDDER.DRL"       PRG
33 BLOCKS USED.
READY.
```

11.4 / – load BASIC program

Loads a BASIC file, like ‘LOAD"FILE",⟨*current device*⟩’.

Example:

To load a program from directory listing:


```
@$

1 "TEST                " 10,64
0    "PLUGINS"           DIR
132  "2 PLANETS BY"      FUN
45   "DRAZPAINT 2.0"     PRG
/1   "DRAZLACE V1.0"     PRG
64   "TASM"              PRG
33   "SHUDDER.DRL"       PRG
315 BLOCKS USED.
READY.

SEARCHING FOR DRAZLACE V1.0
LOADING $0801-$3023
READY.
```

11.5 % – load assembly program

Loads an assembly file, like ‘LOAD"FILE",⟨*current device*⟩,1’, but does not restart program if used in BASIC program.

Example:

To load a program from directory listing:

```
LIST

1 "TEST              " IDE64
0    "PLUGINS"          DIR
132  "2 PLANETS BY"     FUN
45   "DRAZPAINT 2.0"    PRG
41   "DRAZLACE V1.0"    PRG
¼4   "TASM",8           PRG
33   "SHUDDER.DRL"      PRG
315 BLOCKS USED.
READY.

SEARCHING FOR TASM
LOADING $9000-$CF00
READY.
```

11.6 ´ – verify assembly file

Verifies an assembly file, like ‘VERIFY"FILE",⟨*current device*⟩,1’.

Example:

Verify Drazlace:

```
´DRAZLACE*

SEARCHING FOR DRAZLACE*
VERIFYING $0801-$3023
OK

READY.
```

11.7 ↑ – autostart BASIC program

Loads a BASIC file, like ‘/’, and then starts it with ‘RUN’.

Example:

Load Drazlace and start it:

```
↑DRAZLACE*

SEARCHING FOR DRAZLACE*
LOADING $0801-$3023
READY.
R
```

11.8 ← – save BASIC program

Saves a BASIC file, like '`SAVE"FILE"`,⟨*current device*⟩'.

Example:

To save a program:

```
←MYFILE

SAVING MYFILE $0801-$0932
READY.
```


11.9 £ – autostart assembly program

Loads an assembly file, like '%', and then starts it with `JMP` at it's load address.

Example:

Load and start TASM:

```
£TASM

SEARCHING FOR TASM
LOADING $9000-$CF00
```

11.10 . – change directory

Changes directory, like ‘CD’, but much better when used on directory list.

Example:

Enter directory ‘PLUGINS’:

```
@$

1 "TEST              " IDE64
.    "PLUGINS"           DIR
132  "2 PLANETS BY"      FUN
45   "DRAZPAINT 2.0"     PRG
41   "DRAZLACE V1.0"     PRG
64   "TASM"              PRG
33   "SHUDDER.DRL"       PRG
315 BLOCKS USED.
READY.

READY.
```

11.11 # – execute shell

Loads the machine language program called ‘1//:SH’ from the system drive and executes it. The last used device number will be placed at $FF. The pointer $7A points to the rest of the non-tokenized command line.

The assembly program in Listing 12 defines ‘#T’ to start Turbo Assembler from ‘1//UTIL/:TASM’ from the system drive.

```
        *=$334

        jsr $79
        cmp #"t"        ;#T?
        bne not
        ldx $ba         ;boot device
```

```
        ldy #1          ;,1
        jsr $ffba       ;setlfs
        lda #nameend-name
        ldx #<name
        ldy #>name
        jsr $ffbd       ;setnam
        lda #0
        jsr $ffd5       ;load it!
        bcs not         ;error?
        lda $ff
        sta $ba         ;restore last used device
        jmp $9000       ;start tasm

not     lda $ff
        sta $ba         ;restore last used device
        jmp $e37b       ;exit with ready.

name    .text "1//util/:tasm"
nameend
```

Listing 12: Shell sample source

12 BASIC extensions

On the following few pages the new and changed BASIC commands are described. In the format descriptions everything between '[' and ']' is optional, and '⟨*name*⟩' means a parameter.

As a bugfix the ?SYNTAX ERROR bug of LIST command was fixed, listing of BASIC programs protected by 'REM L' is not a problem anymore.

Binary, octal, decimal and hexadecimal numbers are supported in expressions:

```
PRINT %11; &11; 11; $11
 3  9  11  17

READY.
```

The BASIC editor was extended with redefinable function keys and additional editor key combinations. The complete list is in Table 15.

To enable or disable function keys see the setup option "3.1.12 Function keys".

Some of the functions are automatically disabled when in program mode, or when the screen is memory relocated. This is to prevent bugs and unexpected behaviour.

12.1 CD – change directory

Sends change directory to the last used or specified device. (Same as '@CD:PARAMETER') There's no short form of this command.

Format:

CD"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*directory name*⟩"[,⟨*device #*⟩]

Example:

```
CD"GAMES"
READY.
```

12.2 CDCLOSE – insert medium

Sends close tray command to the last used or specified device. (Same as '@U0>E0') Not very useful, as the tray will be automatically inserted on first medium access anyway. The short form is 'CDCLO'.

Format:

CDCLOSE [⟨*device #*⟩]

12.3 CDOPEN – eject medium

Sends eject medium command to the last used or specified device. Beside CD-ROM and DVD drives it works with LS-120 or Zip drive too. (Same as '@U0>E1') The short form is 'CDOP'.

Format:

CDOPEN [⟨*device #*⟩]

12.4 CHANGE – change device number

Sends change device number to the last used or specified device. (Same as '@S-8' or '@S-D') The short form is 'CHA'.

Format:

CHANGE [⟨*device #*⟩]

Example:

Change device 12 to be device 8:

```
CHANGE12
READY.
```

Change device 8 back:

```
CHANGE8
READY.
```

12.5 DATE – display date

Prints the date and time[10] of the last used or specified device. (Same as '@T-RA') There's no short form of this command.

Format:

DATE [⟨*device #*⟩]

Example:

```
DATE
THUR 08/12/04 08:51:18 PM
READY.
```

[10] For date format see "15.4.8 Reading time from RTC".

12.6 DEF – redefine F-keys

If you do not like the default F1–F8 function key assignment, then this command can change it. The best practice to make this permanent to put it into the boot file, and select "Power on" or "Always" in the setup for "3.1.2 Start boot file". Of course the "3.1.12 Function keys" setting must be enabled to make use of the function keys. All 8 string parameters are mandatory, if the string contains a CHR$(13) character then a RETURN key press is simulated.

Format:

DEF"⟨*F1*⟩","⟨*F3*⟩","⟨*F5*⟩","⟨*F7*⟩","⟨*F2*⟩","⟨*F4*⟩","⟨*F6*⟩","⟨*F8*⟩"

```
10 F1$="%0:*":F2$="/0:*":F3$="@$:*"+CHR$(13)
20 F4$="":F5$="LIST"+CHR$(13):F6$="SYS."+CHR$(13)
30 F7$="RUN:"+CHR$(13):F8$="MAN"+CHR$(13)
40 DEF F1$,F3$,F5$,F7$,F2$,F4$,F6$,F8$
50 PRINT CHR$(145)"    "CHR$(145)CHR$(145);
60 NEW
```

Listing 13: This boot file redefines the F-keys, and cleans up the screen

12.7 DIR – list directory

Lists the directory of the last used or specified device. (Same as '@$:PATTERN') There's no short form of this command.

Format:

DIR["[[⟨*part #*⟩][⟨*path*⟩]:]⟨*pattern*⟩"[,⟨*device #*⟩]]

Example:

```
DIR"*=B
```

```
3 "ACE                 " 1DE64
0     "BIN"                DIR
0     "ETC"                DIR
0     "HOME"               DIR
0 BLOCKS USED.
READY.
```

12.8 HDINIT – redetect drives

Trys to auto detect drives connected to the cartridge. The short form is 'HD'.

Format:

HDINIT

Example:

```
HDINIT
#1:ST9385AG
#2:UPCLINK

READY.
```


12.9 INIT – init memory

Fills memory with nulls or with the specified byte, and then it performs a reset. Useful before linking. The short form is 'INI'.

Format:

INIT [⟨*fill byte #*⟩]

Example:

Fill memory with $55

```
INIT$55
```

12.10 KILL – disable cartridge

Switches cartridge off. Useful if you suspect compatibility problems with a program. Will also shut down power managed drives, if typed as 'KILL!'. The short form is 'KI'.

Format:

KILL[!]

Example:

```
KILL
BYE!

READY.
```

12.11 KILLNEW – recover basic program

This command recovers "lost" basic program after a NEW or a reset, as long as the memory is still intact. (no new variables were created) There's no short form of this command.

Format:

KILLNEW

Example:

```
KILLNEW

READY.
```

12.12 LL – long directory list

Pretty verbose directory list for power users. First line is the directory label, same parameters as for command DIR. No short form.

Format:

LL["[[⟨*part #*⟩][⟨*path*⟩]:]"[,⟨*device #*⟩]]

Examples:

Simple listing:

```
LL

D--RWX-              1980-01-01 00:00:00
"WORK             " DIR 560/5/36
DC-R-XH              0 1980-01-01 00:00:00
"%DELETED  FILES%" DIR 560/5/35
-CDRWX-            363 2004-08-12 16:12:13
"LIST"               TXT 568/12/4
-CDRWX-         820528 2004-08-11 16:32:26
"UTILITIES"          ZIP 560/5/37
-CDRWX-        2234108 2004-08-11 18:46:01
"TXT"                ZIP 562/9/21
-CDRWX-          25513 2004-08-11 19:29:43
"CHANNEL15"          ASM 568/11/9
RCDRW--          32002 2004-09-08 14:33:39
"USERS"              REL 570/4/3 127
READY.
```

Redirect listing into file for later review.

```
OPEN 1,12,1,"LIST,S":CMD 1:LL:PRINT#1:CLOSE 1

READY.
```

Listing works on 1541, 1570, 1571, 1581, and CMD drives (on all non-extended native partitions) too:

```
LL"",8

"GEOS 128 V2.0AM " CP 2A
P--                2 1988-08-22 13:00
```

Part		Meaning
-	. - R D L	DEL entry Normal file Relative file Directory Link
C	C	Closed
D	D	Deletable
R	R	Readable
W	W	Writable
X	X	Loadable
-	H	Hidden
363		Size of file in bytes (block for non-IDE64)
2004-08-12		Date
16:12:13		Time
"LIST"		Filename
TXT		File type
568/12/4		Disk address (CHS or LBA)
		Record length (only for relative files)

Table 14: LL list format

```
"GEOS"              PRG 19/15
P--                   6 1988-08-22 13:00
"GEOS BOOT"         PRG 19/17
U--                 156 1988-08-22 13:00
"GEOS KERNAL"       USR 19/18
U-W                 137 1988-10-10 13:06
"128 DESKTOP"       USR 8/20
U-W                  79 1988-09-08 16:50
"128 CONFIGURE"     USR 15/6
R-W                  13 2000-00-00 00:00
"DATA"              REL 10/6 76 9/11
BREAK
READY.
```

12.13 LOAD – load a program

Loads a program file into memory. Using no device number selects last used device. No filename means '∗', except for tape. The short form is 'LO'.

Format:

LOAD["[[⟨*part #*⟩][⟨*path*⟩]:]⟨*filename*⟩"[,⟨*device #*⟩[,⟨*mode #*⟩]]]

Example:

```
LOAD

SEARCHING FOR *
LOADING $0801-$099E
READY.
```

12.14 MAN – start manager

Starts the built in file manager. To force the re-read of 'MAN,USR' start the manager with 'MAN!'. The short form is 'MA'.

Format:

MAN[!]

12.15 MKDIR – create directory

Sends make directory to the last used or specified device. (Same as '@MD:NEWDIR') The short form is 'MK'.

Format:

MKDIR"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*directory name*⟩"[,⟨*device #*⟩]

Example:

```
MKDIR"PICS",12
READY.
```

12.16 RM – remove file

Sends scratch file to the last used or specified device. (Same as '@S:FILENAME') There's no short form of this command.

Format:

RM"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*file name*⟩"[,⟨*device #*⟩]

Example:

```
RM"OLDSTUFF",8
READY.
```

12.17 RMDIR – remove directory

Sends remove directory to the last used or specified device. (Same as '@RD:DIRNAME') There's no short form of this command.

Format:

RMDIR"[[⟨*part #*⟩][⟨*path*⟩]:]⟨*directory name*⟩"[,⟨*device #*⟩]

Example:

```
RMDIR"OLDDIR"
READY.
```

12.18 SAVE – save a program

Saves program to disk. Using no device number selects last used device. You'll get a '?FILE DATA ERROR' if an error happens during save. The short form is 'sA'.

Format:

SAVE"[[@][⟨*part #*⟩][⟨*path*⟩]:]⟨*filename*⟩"[,⟨*device #*⟩[,⟨*mode #*⟩]]

Example:

```
SAVE"TEST"
SAVING TEST $0801-$1A43
READY.
```


12.19 SYS – start ML program

This keyword can be used to call machine code subroutines for BASIC. Optionally the accumulator, the X and Y registers and the status can be specified separated by semicolons.

Format:

SYS ⟨*address #*⟩[;⟨*a #*⟩[;⟨*x #*⟩[;⟨*y #*⟩[;⟨*sr #*⟩]]]]

Examples:

Print a star to the center of screen:

```
SYS $FFF0;;12;20;0:PRINT"*"
```

Start the built in monitor:

```
SYS.
```

12.20 VERIFY – verify program

Verifies that the program in memory matches the on-disk version. Using no device number selects last used device. No filename means '∗'. The short form is 'vE'.

Format:

VERIFY["[[⟨*part #*⟩][⟨*path*⟩]:]⟨*filename*⟩"[,⟨*device #*⟩[,⟨*mode #*⟩]]]

Example:

```
VERIFY"TEST"

SEARCHING FOR TEST
VERIFYING $0801-$1A43
OK

READY.
```

Key	Result	Function
F1	↑!∗,P RETURN	LOAD"!∗,P" and RUN
F3	@$ RETURN	List directory
F5	LI RETURN	List program
F7	RU: RETURN	Run program
F2	%:∗ RETURN	LOAD":∗",DR,1
F4	@$∗=P	List programs in directory
F6	LL RETURN	Detailed directory list
F8	MA RETURN	Start manager
DEL	Editing	Delete and to end of line
CONTROL + ←	Function change	Toggle SCPU turbo mode
CONTROL + A	Cursor move	To start of line
CONTROL + D	Device number	Change last used device
CONTROL + E	Cursor move	To end of line
CONTROL + I	Cursor move	To next tab position
CONTROL + J	Cursor move	To next line
CONTROL + K	Editing	Clear rest of line
CONTROL + L	Editing	Clear line
CONTROL + R	Function change	Toggle key repeat
CONTROL + U	Editing	Delete to start of line
CONTROL + V	Screen	Initialize VIC II
CONTROL + X	Editing	Delete char at cursor
C= + INS	Function change	Toggle insert/overwrite
C= + RETURN	Cursor move	Skip lines starting on left

Table 15: Default BASIC function and editor keys

12

13 Programming in assembly

13.1 Standard KERNAL routines

These routines work with all types of drives, you should use them in your programs for compatibility with IDE64, RamLink and other non-serial bus drives.

13.1.1 READST – read status byte ($FFB7)

Implementation:	Standard KERNAL
Communication registers:	None
Preparatory routines:	None
Error returns:	None
Status:	None
Registers affected:	A

Returns the status in accumulator. It's used to detect errors, end of file, etc.

```
        jsr chrin       ;read data from file
        sta data,y
        jsr readst      ;test status
        and #$40        ;end of file flag
        bne endoffile
```

Listing 14: Check end of file while reading

Bit	Meaning
7	Device not present
6	End of File
5	(Tape CRC error)
4	Verify/read error (Tape read error)
3	(Tape long block)
2	(Tape short block)
1	Timeout on receive
0	Timeout on send

Table 16: Device status ($90)

13.1.2 SETMSG – control KERNAL messages ($FF90)

Implementation: Standard KERNAL
Communication registers: A
Preparatory routines: None
Error returns: None
Status: None
Registers affected: A

Controls IDEDOS and KERNAL message printing. Messages are things like 'SEARCHING FOR XXX' and 'I/O ERROR#05'. Sometimes it's useful to suppress such messages to not destroy the screen.

```
        lda #$00
        jsr setmsg      ;turn off messages
```

Listing 15: Turn off messages to prevent screen distortion

Bit	Meaning
7	Full error messages (LOADING, etc.)
6	KERNAL error messages (I/O ERROR#x)
5–0	Undefined

Table 17: Messages ($9D)

13.1.3 STOP – scan stop key ($FFE1)

Implementation:	Standard KERNAL
Communication registers:	A
Preparatory routines:	None
Error returns:	None
Status:	None
Registers affected:	A, X

Checks if STOP was pressed, and calls CLRCHN if so.

```
        jsr chrin
        sta ($fb),y
        jsr stop
        beq stopped     ;stop was pressed
        ...
stopped lda #filenum
        jmp close       ;close file and exit
```

Listing 16: Check STOP key while reading a file

13.1.4 SETLFS – set file parameters ($FFBA)

Implementation: Standard KERNAL
Communication registers: A, X, Y
Preparatory routines: None
Error returns: None
Status: None
Registers affected: None

Sets logical file number, device number and secondary address. These parameters are the same as for the OPEN BASIC command. It's used before OPEN, LOAD and SAVE. File number is ignored for LOAD and SAVE, and secondary address too for SAVE.

File number	Meaning
0	Illegal
1–127	Nothing special
128–255	BASIC adds CHR$(10) after each line

Table 18: File numbers ($B8)

Device #	Device	Device #	Device
0	Keyboard	6–7	IEC bus plotters
1	Datassette	8–11	IEC bus disk drives
2	RS-232C device	12–30	IEC bus other
3	Display	8–22	IDEDOS drives
4–5	IEC bus printers	31–255	Illegal

Table 19: Device numbers ($BA)

Secondary address	Open mode
0	Read access (LOAD)
1	Write access (SAVE)
2–14	Bi-directional data channel
15	Status and command channel
16–127	Illegal
128–255	No secondary address

Table 20: Secondary addresses ($B9)

13.1.5 SETNAM – set filename ($FFBD)

Implementation: Standard KERNAL
Communication registers: A, X, Y
Preparatory routines: None
Error returns: None
Status: None
Registers affected: None

Sets filename and name length for OPEN, LOAD, and SAVE routines.

NOTE

The filename must be located below $D000 in memory. Don't forget to set $01 if it's under the BASIC ROM before calling OPEN, LOAD or SAVE!

13.1.6 OPEN – open file ($FFC0)

Implementation:	IDEDOS Extended
Communication registers:	None
Preparatory routines:	SETLFS, SETNAM
Error returns:	1, 2, 4, 5, 6, 7, 240 (see Table 21)
Status:	$00, $80
Registers affected:	A, X, Y

Opens a file and associates it with a logical file number.

```
        lda #filenum    ;file number (1-255)
        ldx $ba         ;actual device number
        ldy #secaddy    ;secondary address
        jsr setlfs
        lda #8          ;filename length
        ldx #<name      ;address low byte
        ldy #>name      ;high byte
        jsr setnam
        ...
        jsr open
        ...
name    .text "filename"
```

Listing 17: Set file parameters and filename for OPEN

13.1.7 CLOSE – close file ($FFC3)

Implementation:	IDEDOS Extended
Communication registers:	A
Preparatory routines:	None
Error returns:	0, 240 (see Table 21)
Status:	$00, $03, $80
Registers affected:	A, X, Y

Closes the file associated by the logical file number.

NOTE

Unlike serial devices IDEDOS does not close it's files when closing the error channel.

```
        lda #filenum    ;opened file number
        jsr close
```

Listing 18: Close a file

13.1.8 CHKIN – set file as input ($FFC6)

Implementation:	IDEDOS Extended
Communication registers:	X
Preparatory routines:	valid OPEN
Error returns:	0, 3, 5, 6 (see Table 21)
Status:	$00, $03, $80
Registers affected:	A, X

Set standard input to the logical file number. This means now you can use CHRIN, GETIN and READ on the file.

```
...
jsr open         ;open file

ldx #filenumber ;opened file number
jsr chkin        ;set input

ldx #0
jsr chrin        ;get bytes
sta $400,x
...
```

Listing 19: Start to read from a file

13.1.9 CHKOUT – set file as output ($FFC9)

Implementation:	IDEDOS Extended
Communication registers:	X
Preparatory routines:	valid OPEN
Error returns:	0, 3, 5, ~~7~~ (see Table 21)
Status:	$00, $03, $80
Registers affected:	A, X

Set standard output to the logical file number. This means now you can use CHROUT and WRITE on the file.

```
...
jsr open        ;open file

ldx #filenumber ;opened file number
jsr chkout      ;set output

ldx #0
lda $400,x
jsr chrout      ;write out data
...
```

Listing 20: Starts to write to a file

13.1.10 CHRIN – input character ($FFCF)

Implementation:	IDEDOS Extended
Communication registers:	None
Preparatory routines:	valid OPEN, CHKIN
Error returns:	None
Status:	$00, $40, $42, $52, $80
Registers affected:	A (Y but not for file I/O)

Get a character from standard input. If it's the screen the cursor will appear and you can type in characters until RETURN.

```
...
jsr chkin

ldy #0
jsr chrin
sta data,y
iny
...
```

Listing 21: Read in a byte from a file

13.1.11 GETIN – get character ($FFE4)

Implementation:	IDEDOS Extended
Communication registers:	None
Preparatory routines:	valid OPEN, CHKIN
Error returns:	None
Status:	$00, $40, $42, $52, $80
Registers affected:	A (X, Y but not for file I/O)

Get a character from standard input. If it's the screen the last pressed keys from the keyboard buffer will be be returned. If there is none, then $00 will be returned.

```
        ...
        jsr chkin

        ldy #0
        jsr getin
        sta data,y
        iny
        ...
```

Listing 22: Read in a byte from a file

```
        ...
        jsr clrchn      ;keyboard/screen
        ...
wait    jsr getin       ;get key
        beq wait        ;nothing pressed?
```

Listing 23: Wait until a key is pressed

13.1.12 CHROUT – output character ($FFD2)

Implementation:	IDEDOS Extended
Communication registers:	A
Preparatory routines:	valid OPEN, CHKOUT
Error returns:	0 (see Table 21)
Status:	$00, $03, $80
Registers affected:	None

Output a character to standard output.

```
...
jsr chkout

lda #$00
jsr chrout      ;write 0
...
```

Listing 24: Write a byte to a file

13.1.13 CLALL – close all files ($FFE7)

Implementation:	IDEDOS Extended
Communication registers:	None
Preparatory routines:	None
Error returns:	None
Status:	$00, $03, $80
Registers affected:	A, X

Forget about all files and set standard input and output to keyboard and screen.

NOTE

In it's standard implementation this call wipes out the open files table without closing the files for real. You should close files by using the CLOSE call, as this call is only intended to be used at the beginning of programs to make sure all files are closed. However IDEDOS will close it's own files to reduce the chance of creating splat files.


```
        jsr clall       ;close files, default I/O
        jmp run         ;run program
```

Listing 25: Make sure all files are closed before starting

13.1.14 CLRCHN – reset input and output ($FFCC)

Implementation:	IDEDOS Extended
Communication registers:	None
Preparatory routines:	None
Error returns:	None
Status:	$00, $03, $80
Registers affected:	A, X

Set standard input and output to keyboard and screen.

```
...
jsr chkin
...
jsr clrchn      ;set default I/O
lda #1
jsr close
```

Listing 26: Set standard keyboard and screen for in and output

13.1.15 LOAD – load ram from file ($FFD5)

Implementation:	IDEDOS Extended
Communication registers:	A, X, Y
Preparatory routines:	SETLFS, SETNAM
Error returns:	0, 4, 5, 8, 9, 16 (see Table 21)
Status:	$00, $10, $40, $42, $50, $52, $80
Registers affected:	A, X, Y

Loads or verifies a program file. Program files start with a 2 byte little endian memory start address.

NOTE

The standard implementation does not permit to load below the I/O area. However IDEDOS switches $01 memory configuration register automatically if needed to allow of loading huge programs. Verifying RAM under I/O is unsupported.

```
        lda #1          ;filename length
        ldx #<dirnam
        ldy #>dirnam    ;filename pointer
        jsr setnam
        lda #1          ;file number
        ldx $ba         ;actual device number
        ldy #0          ;sec.address 0=specified,
        jsr setlfs      ;else original location

        lda #$00        ;load flag (1=verify)
        ldx #<dirbuff
        ldy #>dirbuff   ;new start address
```

```
        jsr load
        bcc loadok
        ...
        rts

loadok  stx $ae         ;new end after load/verify
        sty $af
        ...
        rts

dirnam  .text "$"
```

Listing 27: Load the program formatted directory listing to “dirbuff”

13.1.16 SAVE – save ram to file ($FFD8)

Implementation:	IDEDOS Extended
Communication registers:	A, X, Y
Preparatory routines:	SETLFS, SETNAM
Error returns:	0, 5, 8, 9, 24 (see Table 21)
Status:	$00, $03, $80
Registers affected:	A, X, Y

Save program to file. A 2 byte start address is inserted in front of the data.

NOTE

It's possible to save RAM from $0200 to $CFFF. For saving RAM under the BASIC ROM don't forget to set $01!

```
databegin = $fb

        lda #1          ;file number
        ldx $ba         ;actual device number
        ldy #0          ;sec.address
        jsr setlfs
        lda #8          ;filename length
        ldx #<name
        ldy #>name      ;filename pointer
        jsr setnam

        lda #<$1000
        sta databegin   ;begin
        lda #>$1000
        sta databegin+1
```

```
        lda #databegin
        ldx #<$8000
        ldy #>$8000      ;end
        jsr save
        ...
name    .text "filename"
```

Listing 28: Save the memory $1000–$7FFF to a file

Accu	Meaning
0	Routine terminated by the STOP key
1	Too many open files
2	File already open
3	File not open
4	File not found
5	Device not present
6	File is not an input file
7	File is not an output file
8	File name is missing
9	Illegal device number
16	Out of memory (LOAD)
24	File data error (SAVE)
240	Top-of-memory change RS-232 buffer (de)allocation

Table 21: Error codes returned by IDEDOS and KERNAL

13.2 IDE64 specific routines

13.2.1 IDE64 card detection

If you want that your application using READ or WRITE calls remain still runnable on a not IDE64 equipped machine, check for IDE64 presence before calling these two calls, and use standard routines instead. (Imagine what happens at JSR $DEF1 if there's open I/O space at $DE00–$DEFF...)

```
        lda $de60       ;check IDE64
        cmp #$49
        bne old
        lda $de61
        cmp #$44
        bne old
        lda $de62
        cmp #$45
        bne old

        lda #zp
        jsr $def1
        bcs old2        ;write not available?
        rts

old2    ldx #channel
        jsr chkout      ;or chkin
old     ...             ;old byte by byte routine
        rts
```

Listing 29: Detect IDE64 before write and workaround if not present

This works nice, unless someone has an old ActionReplay installed instead, which will crash...

```
        lda $df09
        cmp $df09
        bne notaction   ;no action replay there
        cmp #$78        ;maybe it's and AR
        beq old         ;do not check for IDE64

notaction
        ...             ;IDE64 detection as above

old     ...             ;old byte by byte routine
        rts
```

Listing 30: Check for ActionReplay first to avoid crash

13.2.2 WRITE – write ram ($DEF1)

Implementation:	IDEDOS only
Communication registers:	A, X, Y
Preparatory routines:	valid OPEN, CHKOUT
Error returns:	5, 7, 9, 24 (see Table 21)
Status:	$00, $80
Registers affected:	A, X, Y

Save memory to an IDE64 drive. It's much faster than calling CHROUT a lot of times.

To avoid possible compatibility problems make sure that there's an IDE64 installed (13.2.1 IDE64 card detection), and consider in-

cluding the byte-by-byte replacement, as shown in "13.2.3 WRITE – replacement"!

NOTE

It's not possible to save under I/O. (e.g. saving from $D800 will save color RAM) To access RAM under the BASIC and KERNAL ROM, set $01 correctly. Saving RAM under the KERNAL ROM is not supported on SuperCPU.

13.2.3 WRITE – replacement

Here's an example replacement byte-by-byte routine for non-IDE64 drives, can be shortened of course:

```
        stx wnum
        sty wnum+1      ;byte to be written
        stx tmp
        sty tmp+1
        tax
        lda $00,x
        sta pointer
        lda $01,x
        sta pointer+1   ;copy pointer

        ldx #channel
        jsr chkout      ;do select output
        bcs end         ;error happened

loop    lda wnum        ;write loop
        ora wnum+1
        beq end
```

```
        ldy #0
        lda (pointer),y
        jsr chrout
        lda $90         ;status
        bne end2        ;error during write

        lda wnum
        bne at2
        dec wnum+1
at2     dec wnum

        inc pointer
        bne loop
        inc pointer+1
        jmp loop

end2    jsr clrchn
        lda #5          ;device not present
        sec
end     php
        pha
        lda tmp
        sec
        sbc wnum
        tax
        lda tmp+1
        sbc wnum+1      ;calculate
        tay             ;bytes written
        pla
        plp
```

```
        rts
```

Listing 31: An compatibility fallback in case WRITE is not available

13.2.4 READ – read ram ($DEF4)

Implementation:	IDEDOS only
Communication registers:	A, X, Y
Preparatory routines:	valid OPEN, CHKIN
Error returns:	5, 6, 9, 24 (see Table 21)
Status:	$00, $40, $42, $52, $80
Registers affected:	A, X, Y

Load data block from a IDE64 drive. Much faster than a lot of CHRIN or GETIN.

To avoid possible compatibility problems make sure that there's an IDE64 installed (13.2.1 IDE64 card detection), and consider including the byte-by-byte replacement, as shown in "13.2.3 WRITE – replacement"!

NOTE

This routine switches $01 automatically when the I/O area is reached, to load to the RAM below. To read into color RAM ($D800–$DBFF) the read must start within this area. No other I/O areas are supported.

```
        lda #1          ;source file number
        ldx $ba         ;actual device number
        ldy #0          ;secondary address, read
```

```
        jsr setlfs
        lda #outputname-inputname
        ldx #<inputname
        ldy #>inputname
        jsr setnam
        jsr open        ;open input file

        lda #2          ;destination filenumber
        ldx $ba         ;actual device number
        ldy #1          ;secondary address, write
        jsr setlfs
        lda #status-outputname
        ldx #<outputname
        ldy #>outputname
        jsr setnam
        jsr open        ;open output file

        lda #<startadd
        sta $fb
        lda #>startadd  ;buffer start address
        sta $fc

        ldx #1          ;set input to source file
        jsr chkin

        ldx #2
        jsr chkout      ;set output to dest. file

loop    lda #$fb        ;start address is here
        ldx #<blocksize
        ldy #>blocksize ;block size
```

```
        jsr read        ;read
        bit $90         ;readst
        php             ;status to stack

        lda #$fb
        jsr write       ;write
        plp             ;status
        bvc loop        ;test end of file

        lda #2
        jsr close       ;close output file
        lda #1
        jsr close       ;close input file
        jsr clall       ;set default I/O device
        rts

inputname .text "//bin/:input-file"
outputname .text "//tmp/:output-file"
status  .byte 0
```

Listing 32: Copy a file using READ and WRITE

13.2.5 READ – replacement

Here's an example replacement byte-by-byte routine for non-IDE64 drives:

```
        stx rnum
        sty rnum+1      ;byte to be written
        stx tmp
        sty tmp+1
```

```
        tax
        lda $00,x
        sta pointer
        lda $01,x
        sta pointer+1   ;copy pointer

        ldx #channel
        jsr chkin       ;do select output
        bcs end         ;error happened

loop    lda rnum        ;read loop
        ora rnum+1
        beq end

        jsr chrin
        ldx $90         ;status
        beq ok
        cpx #$40
        bne end2        ;error happened
ok      ldy #0
        sta (pointer),y

        lda rnum
        bne at2
        dec rnum+1
at2     dec rnum

        clc
        txa
        bne end         ;end of file reached
```

```
        inc pointer
        bne loop
        inc pointer+1
        jmp loop

end2    jsr clrchn
        lda #5          ;device not present
        sec
end     php
        pha
        lda tmp
        sec
        sbc rnum
        tax
        lda tmp+1
        sbc rnum+1      ;calculate
        tay             ;bytes read
        pla
        plp
        rts
```

Listing 33: An compatibility fallback in case READ is not available

13.3 IDE64 compatible programming

13.3.1 Serial bus specific code

Many programs use serial bus specific KERNAL calls which won't work with IDE64 drives. Such code has to be rewritten to use standard KERNAL calls. Here's a list of problematic routines, and what to do with them:

IECOPEN ($F3D5), IECCLOSE ($F642)

Serial bus file open and close. Can be replaced by OPEN and CLOSE.

LISTEN ($FFB1, $ED0C), then SECOND ($FF93, $EDB9)

Used to prepare the device to send data to a channel, can be replaced by CHKOUT.

TALK ($FFB4, $ED09), then TKSA ($FF96, $EDC7)

Used to prepare the device to read data from a channel, can be replaced by CHKIN.

ACPTR ($FFA5, $EE13), CIOUT ($FFA8, $EDDD)

Read and write a byte from and to the serial bus, can be replaced by CHRIN and CHROUT.

UNTLK ($FFAB, $EDEF), UNLSN ($FFAE, $EDFE)

Send untalk and unlisten, can be replaced by CLRCHN.

An example serial bus code fixing can be seen in Listing 37 and Listing 38.

13.3.2 Vector table restoring

A lot of programmers use calls such as `JSR $FF8A` or `JSR $FD15` to restore the kernal IO vectors on page 3 for some reason. Of course this means the program has no more access to IDE64 drives, even if using standard KERNAL calls.

If the vector restore is at the begining of a program, it can be most likely safely removed.

Sometimes it's used to restore the IRQ and NMI vectors, and a simple remove will break the program. See Listing 34 for vector table restore.

```
restor  ldy #$05
lp      lda $fd30,y     ;restore interrupt
        sta $314,y      ;vectors from kernal
        dey
        bpl lp
        rts
```

Listing 34: Interrupt vector restoring

This will restore BRK and NMI too, so the monitor won't work. If the program does not use NMI and you need the monitor for debugging, a simpler patch like in Listing 35 will do.

```
restor  lda #<$ea31
        sta $314        ;restore irq only
        lda #>$ea31
        sta $315
        rts
```

Listing 35: IRQ vector restoring

Some programs really destroy the vector table, in this case one first has to save the vector table at the program's beginning to a safe place, and restore it later. Here's a short example code for vector table handling in Listing 36.

NOTE

Never save a fixed vector table into the program, it won't work without IDE64 or with future versions of IDEDOS!

```
save    sec             ;save vectors on start
        .byte $24       ;skip clc (bit)
restore clc             ;restore them later
        ldx #<safeplace
        ldy #>safeplace
        jmp $ff8d       ;vector

safeplace               ;32 bytes free space
```

Listing 36: Vector save and restore

13.3.3 Direct KERNAL calls

As IDEDOS uses the vector table at $03xx, directly calling KERNAL will skip this indirection, and as a result IDE64 drives cannot be accessed. Table 22 contains the routine addresses to be looking for.

```
    lda #8    ;drive 8
    sta $ba
    lda #$6f  ;channel
    sta $b9
    lda #0
    sta $90
    jsr $ffbd ;setname
    jsr $f3d5 ;open
```

```
    lda #2    ;filenumber
    ldx $ba   ;last drive
    ldy #15   ;channel
    jsr $ffba ;setparam
    lda #0
    jsr $ffbd ;setname
    jsr $ffc0 ;open
```

Old	Replace with	Name
$F34A	$FFC0	OPEN
$F291	$FFC3	CLOSE
$F20E	$FFC6	CHKIN
$F250	$FFC9	CHKOUT
$F157	$FFCF	CHRIN
$F13E	$FFE4	GETIN
$F1CA	$FFD2	CHROUT
$F4A5	$FFD5	LOAD
$F5ED	$FFD8	SAVE
$F32F	$FFE7	CLALL
$F333	$FFCC	CLRCHN

Table 22: Direct KERNAL call replacement table

```
lda $ba                          ldx #2    ;filenumber
jsr $ed0c ;listen                jsr $ffc9 ;chkout
lda $b9
jsr $edb9 ;second

lda #$49                         lda #$49
jsr $eddd ;send                  jsr $ffd2 ;chrout

jsr $edfe ;unlisten              jsr $ffcc ;clrchn

lda $ba                          ldx #2    ;filenumber
jsr $ed09 ;talk                  jsr $ffc6 ;chkin
lda $b9
jsr $edc7 ;tksa
```

```
lp  jsr $ee13 ;read
    jsr $ffd2 ;print
    bit $90
    bvc lp

    jsr $edef ;untalk
    jsr $f642 ;close
```

```
lp  jsr $ffcf ;read
    jsr $ffd2 ;print
    bit $90
    bvc lp

    jsr $ffcc ;clrchn
    lda #2
    jsr $ffc3 ;close
```

Listing 37: Serial bus error channel reading replacement

```
    lda #$24  ;$ sign
    sta $fb   ;name
    lda #0
    sta $90
    lda #8    ;drive 8
    sta $ba
    lda #$60  ;channel
    sta $b9
    ldx #$fb
    ldy #0
    lda #1
    jsr $ffbd ;setname

    jsr $f3d5 ;open

    lda $ba
    jsr $ffb4 ;talk
    lda $b9
    jsr $ff96 ;tksa
```

```
    lda #$24  ;$ sign
    sta $fb   ;name

    lda #2    ;filenum
    ldx $ba   ;last drv
    ldy #0    ;channel
    jsr $ffba ;setparam
    lda #$fb
    ldy #0
    lda #1
    jsr $ffbd ;setname

    jsr $ffc0 ;open

    ldx #2    ;filenum
    jsr $ffc6 ;chkin
```

```
    lda #$00
    sta $90
    ldy #3

lp  sty $fb
    jsr $ffa5
    sta $fc
    ldy $90
    bne eof

    jsr $ffa5
    ldy $90
    bne eof
    ldy $fb
    dey
    bne lp
    ldx $fc
    jsr $bdcd ;number
    lda #$20
    jsr $ffd2

lp2 jsr $ffa5
    ldx $90
    bne eof
    tax
    beq eol
    jsr $ffd2
    jmp lp2

eol lda #13
```

```
    ldy #3

lp  sty $fb
    jsr $ffcf
    sta $fc
    ldy $90
    bne eof

    jsr $ffcf
    ldy $90
    bne eof
    ldy $fb
    dey
    bne lp
    ldx $fc
    jsr $bdcd ;number
    lda #$20
    jsr $ffd2

lp2 jsr $ffcf
    ldx $90
    bne eof
    tax
    beq eol
    jsr $ffd2
    jmp lp2

eol lda #13
```

```
    jsr $ffd2                       jsr $ffd2
    ldy #2                          ldy #2
    bne lp                          bne lp

eof                             eof jsr $ffcc ;clrchn

    jsr $f642 ;close                lda #2
                                    jsr $ffc3 ;close
```

Listing 38: A classic serial bus specific directory lister routine. The right column shows the standard KERNAL version

13.3.4 RAM locations

Some important I/O RAM locations are listed here, mostly focused on using IDEDOS, IEC devices or IDE64 services. It's a good idea to backup them if the zero page (or the memory above) is heavily used. Either KERNAL or IDEDOS is using them as input, output or as temporary storage, or they are jump vectors.

$90 *Status bits*
Used by most of the calls for status report. For bit assignment see Table 16. Calls like `CHRIN` rely on the previous value.

$91 *Stop flag*
Used by `LOAD` and `SAVE` for detecting user interruption.

$93 *Verify flag*
Used and updated by `LOAD` for selecting load or verify operation, where 0 means load, anything else verify.

$94 *Serial buffer flag*
Bit 7 set if there's a buffered character. Used by IEC routines.

$95 *Serial output buffer*
Buffered character for IEC bus before send. Used by IEC routines.

$98 *Open files*
Contains the number of open files, not more than 10. Updated by `OPEN` and `CLOSE`. Also used by `LOAD` and `SAVE` in IDEDOS.

$99 *Input device*
Actual input device. Default is 0 (keyboard). Do not modify it directly, read only. Set by `CHKIN`, reset by `CLRCHN` or by any error condition. Also reset by `LOAD` and `SAVE` in IDEDOS.

$9A *Output device*
Actual output device number. Default is 3 (screen). Do not modify it directly, read only. Set by `CHKOUT`, reset by `CLRCHN` or any error condition. Also reset by `LOAD` and `SAVE` in IDEDOS.

$9D *Message mode*
Flag for KERNAL error messages. $00 no messages, $40 error numbers only, $80 all errors. Could cause unexpected messages on error if not set right.

$A3 *EOI flag*
EOI flag for IEC bus. Used by IEC routines.

$A4 *Serial input buffer*
Incoming character from IEC bus. Used by IEC routines.

$AC–$AD *End of save pointer*
Used by SAVE.

$AE–$AF *End of program pointer*
Used by LOAD and SAVE.

$B7 *File name length*
File name length for OPEN, LOAD, SAVE.

$B8 *Logical file number*
Logical file number for OPEN, LOAD, SAVE. Updated by CLOSE, CHKIN and CHKOUT as well.

$B9 *Secondary address*
Secondary address for OPEN, LOAD, SAVE. Updated by CLOSE, CHKIN and CHKOUT as well.

$BA *Device number*
Device number for OPEN, LOAD, SAVE. Updated by CLOSE, CHKIN and CHKOUT as well.

$BB–$BC *File name pointer*
Pointer to the current file name for OPEN, LOAD, SAVE.

$C1–$C2 *Start of save pointer*
Used by SAVE.

$259–$262 *Logical file number table*
Table for open files, logical file numbers. Updated by OPEN and CLOSE. Also used by LOAD and SAVE in IDEDOS.

$263–$26C *Device number table*
Table for open files, device numbers. Updated by `OPEN` and `CLOSE`. Also used by `LOAD` and `SAVE` in IDEDOS.

$26D–$276 *Secondary address table*
Table for open files, secondary address numbers. Updated by `OPEN` and `CLOSE`. Also used by `LOAD` and `SAVE` in IDEDOS.

$28F–$290 *Keyboard decode vector*
This jump vector is used for C128 keyboard handling.

$300–$30B *BASIC vectors*
These jump vectors are used for implementing the BASIC extensions.

$316–$317 *BRK vector*
This jump vector is used for breakpoint handling and invokes the monitor.

$318–$319 *NMI vector*
This jump vector is used for monitor and setup keyboard shortcut handling, and proper reset of IDEDOS devices in case of abort.

$31A–$333 *KERNAL vectors*
IDEDOS relies on this jump vector table to implement access to all it's devices.

WARNING!

Do not hardcode any jump vector values, pointers are changed between IDEDOS versions! There are also other cartridge expansions out there which use them...

13

14 PCLink

IDEDOS also supports a special "PCLink" device, which is a network virtual drive. It's mostly implemented on the host computer using a server software called "ideservd".

PCLink is most widely used for file transfers between the C64 and the host computer's filesystem, as the host's files and directories can be directly accessed. Beyond file transfers it also supports all (except direct access) operations which are possible with a normal drive.

14.1 PCLink over IEC bus

The slowest method but the easiest, if you already have a X1541, XE1541, XM1541 or XA1541 cable. All serial bus devices must be switched off or removed from the bus for correct operation. The maximal transfer speed is around $\sim$4.5 kB/s.

Pin	C64 IEC bus	Pin	Host printer port
2	GND	18–25	GND
3	ATN	1	$\overline{\text{STROBE}}$
4	CLK	14	$\overline{\text{AUTOFEED}}$
5	DATA	17	$\overline{\text{SELECT IN}}$

Table 23: X1541 PCLink cable

This cable will not work with all printer ports. The other cable types are described in the Star Commander's documentation, written by Joe Forster/STA.

14.2 PCLink over PC64 cable

This connection is made between the parallel port of the host computer and the user port of a C64 using a cable similar to Laplink. The layout is the same as for the PC64 cable. The maximal transfer speed is around $\sim$9 kB/s.

NOTE

There are no optional wires here, so if you have an old PCLink cable, then check if B and FLAG are connected!

Pin	C64 user port	Pin	Host printer port
A, 1	GND	18–25	GND
B	$\overline{\text{FLAG}_2}$	9	D_7
C	PB_0	15	$\overline{\text{ERROR}}$
D	PB_1	13	SELECT
E	PB_2	12	PAPER
F	PB_3	10	$\overline{\text{ACK IN}}$
H	PB_4	2	D_0
J	PB_5	3	D_1
K	PB_6	4	D_2
L	PB_7	5	D_3
M	PA_2	11	BUSY
N, 12	GND	18–25	GND

Table 24: Parallel PCLink cable

WARNING!

Incorrectly built cable can damage the printer and/or userport! Same for plugging the cable while the computers are turned on...

14.3 PCLink over RS-232C

A null modem cable with handshake is used between the host computer's serial port and a serial adapter[11] attached to the C64. Communication is done at 115.2 kbit/s[12] resulting in an effective transfer speed of around ∼11 kB/s.

9 pin	C64 serial port	9 pin	25 pin	Host serial port
2	RXD	3	2	TXD
3	TXD	2	3	RXD
7	$\overline{\text{RTS}}$	8	5	$\overline{\text{CTS}}$
8	$\overline{\text{CTS}}$	7	4	$\overline{\text{RTS}}$
5	GND	5	7	GND

Table 25: Serial PCLink cable

14.4 PCLink over ethernet

An ethernet crosslink cable is used between the host computer's network card and a network card[13] attached to the C64. Communication

[11] DUART, SwiftLink, Turbo232 and SilverSurfer serial adapters are supported.
[12] SwiftLink is limited to 38.4 kbit/s.
[13] ETH64 (II), (E)TFE and RR-Net network adapters are supported.

is done using UDP/IP packets resulting in an effective transfer speed of around ∼60 kB/s.

Try to avoid using lossy wireless links, as the protocol neither retransmits lost packets, nor checks UDP checksum.

RJ45	C64	Color	RJ45	Host
1	TX+	White and Orange	3	RX+
2	TX−	Orange	6	RX−
3	RX+	White and Green	1	TX+
4		Blue	7	
5		White and Blue	8	
6	RX−	Green	2	TX−
7		White and Brown	4	
8		Brown	5	

Table 26: Ethernet crosslink PCLink cable

14.5 PCLink over USB

It's using a common 5 pin mini-B to A-type USB cable between the host computer's USB port and the IDE64 V4.1 and V4.2 cartridge. The communication chip used is USB 1.1 and 2.0 compatible. Transfer speed is around ∼40 kB/s.

The IDE64 USB interface is suppored by recent operating systems (as it's just a common FTDI serial FIFO). If not, then there's a driver at http://www.ftdichip.com/Drivers/D2XX.htm. Without the driver ideserv might not work properly.

15 Command channel

This section is about the DOS commands known by IDEDOS. Some examples use the DOS wedge like '@ı', of course the '@' at the beginning of line is not part of the command.

In the format descriptions everything between '[' and ']' is optional, and '⟨*name*⟩' means a parameter.

15.1 File management commands

Here only the file management commands are listed, to learn more about files see section "7 Using files".

15.1.1 Position

Seeking is supported in both relative and regular files. The format for a relative file is:

Format:

"P"+CHR$(⟨*channel #*⟩)+CHR$(⟨*record bits 0–7 #*⟩)+
CHR$(⟨*record bits 8–15 #*⟩)+CHR$(⟨*character #*⟩)

The command is slightly different for regular files:

Format:

"P"+CHR$(⟨*channel #*⟩)+CHR$(⟨*position bits 0–7 #*⟩)+
CHR$(⟨*position bits 8–15 #*⟩)+CHR$(⟨*position bits 16–23 #*⟩)+
CHR$(⟨*position bits 24–31 #*⟩)

For compatibility with other systems two more forms are available. These can't be used to position beyond the end of file.

NOTE

These two command interfaces (F-P and F-P:) are not stable (but works as expected), and the format may change in future. (this does not depend on me of course) Avoid the use of these in your programs for now.

Format:

"F-P"+CHR$(⟨*channel #*⟩)+CHR$(⟨*position bits 0–7 #*⟩)+
CHR$(⟨*position bits 8–15 #*⟩)+CHR$(⟨*position bits 16–23 #*⟩)+
CHR$(⟨*position bits 24–31 #*⟩)
"F-P:"; ⟨*channel #*⟩; ⟨*position bits 0–7 #*⟩; ⟨*position bits 8–15 #*⟩;
⟨*position bits 16–23 #*⟩; ⟨*position bits 24–31 #*⟩

Whenever a new position command is issued, it will flush the file's write buffer to disk if it was dirty. It's possible to seek beyond the end of file and write new data there, in this case the file will be extended and those bytes between the old file end and the current position will all become `CHR$(0)`. It's called a "hole", because this part of file does not use any disk space until it's overwritten with useful data.

```
10 OPEN 15,12,15:OPEN 4,12,4,"FILE,L"
20 P$=CHR$(44)CHR$(1)CHR$(10)
30 PRINT#15,"P"CHR$(4)P$
40 GET#4,A$:CLOSE 4:CLOSE 15
```

Listing 39: Seek in a relative file to the 300th record's 10th byte and read it. (counting begins at record 1 and byte 1)

```
10 OPEN 15,12,15:OPEN 4,12,4,"FILE"
20 P$=CHR$(159)CHR$(134)CHR$(1)CHR$(0)
30 PRINT#15,"P"CHR$(4)P$
```

```
40 GET#4,A$:CLOSE 4:CLOSE 15
```

Listing 40: Seek in file to the 100000th byte and read it. (counting begins at byte 0)

15.2 Filesystem management commands

15.2.1 Initialize

Initialize the filesystem. In case of disk change it redetects the filesystem.

Format:

"I[⟨*partition #*⟩]"

Example:

```
OPEN15,8,15,"I":CLOSE15
```

15.2.2 Scratch

Delete a file, or more files. Multiple files are specified by wildcards, or by a colon, which marks the beginning of a new pattern. The exact filetype can be specified by '='. For empty directories use 'RD'! File must have the DELETABLE flag set, and must be on a writable partition in a writable directory.

NOTE

If the filetype is not given, it means any, so watch out! As there's no way to recover a deleted file, first try to list the directory with the pattern to be sure to hit the right files.

Format:

"S[⟨*partition #*⟩]:⟨*file name*⟩[=⟨*file type*⟩][,⟨*file name*⟩[=⟨*file type*⟩]]"
"S[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*file name*⟩[=⟨*file type*⟩]
[,[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*file name*⟩[=⟨*file type*⟩]]"

Examples:

Delete all files in the current working directory:

```
@S:*
01, FILES SCRATCHED,028,000,000,000
```

Delete files called 'FILE' with any type:

```
@S:FILE
01, FILES SCRATCHED,003,000,000,000
```

Delete all files with type 'BAK':

```
@S:*=BAK
01, FILES SCRATCHED,009,000,000,000
```

Delete the file called 'FILE,PRG'

```
@S:FILE=PRG
01, FILES SCRATCHED,001,000,000,000
```

Delete the file called 'FILE,PRG' in partition 3.

```
@S3:FILE=PRG
01, FILES SCRATCHED,001,000,000,000
```

Delete all files with file type 'OLD' and 'BAK' in directory called 'STUFF'.

```
@S/STUFF/:*=OLD,*=BAK
01, FILES SCRATCHED,015,000,000,000
```

Delete all files with file type 'OLD' and 'BAK' in directory called 'STUFF', and everything from 'STUFF/BAK'

```
@S/STUFF/:*=OLD,*=BAK,/STUFF/BAK/:*
01, FILES SCRATCHED,043,000,000,000
```

15.2.3 Rename and move

Rename or move a file or directory. The source and destination file must be on the same partition, on a writable partition and in a writable directory. The filetype can only be changed for regular files. To rename the directory header, use 'R-H', it's described in section "15.6.5 Rename directory header"!

Moving files is not an atomic operation, this means you can lose data if you turn off the computer at the wrong time! Because the filesystem must be consistent at all time, the move operation takes place by first writing out a non-closed version in the destination directory. If this succeeds then the original file gets removed, and finally the destination will be closed. When moving a directory, the first test non-closed entry will be a deleted one.

Format:

"R[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*new file name*⟩[,⟨*file type*⟩]
=[/⟨*path*⟩/:]⟨*old file name*⟩[,⟨*file type*⟩]"

Examples:

Rename the file called 'OLD,PRG' to 'NEW,SEQ'

```
@R:NEW,SEQ=OLD,PRG
00, OK,000,000,000,000
```

Move 'OLD,PRG' from directory 'SOURCE' into directory 'DEST' as 'NEW,SEQ'

```
@R/DEST/:NEW,SEQ=/SOURCE/:OLD,PRG
00, OK,000,000,000,000
```

15.2.4 Lock

Change the protection flags of a file or directory. Locked files cannot be written to or deleted. Write protected directories can't be modified.

Format:

"L[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*file name*⟩[,⟨*file type*⟩]"

The file must be on a writable partition and in a writable directory. If the filetype is not given, it means any. This command operates only on one file at a time.

Example:

Toggles WRITABLE and DELETABLE flags on 'FILE,PRG'

```
@L:FILE,PRG
00, OK,000,000,000,000
```

15.2.5 Hide

Change the hidden flag of a file or directory. Hidden files or directories are not visible in the directory listing, but otherwise there's no difference. Use them with care, hidden files can easily produce directories which look like empty, but cannot be removed, because not all hidden files were removed from them.

NOTE

This command interface is not stable (but works as expected), and the format may change in future. (this does not depend on me of course) Avoid the use of it in your programs for now.

Format:

"EH[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*file name*⟩[,⟨*file type*⟩]"

The file must be on a writable partition and in a writable directory. If the filetype is not given, it means any. This command operates only on one file at a time.

Example:

Toggle the HIDDEN flag on 'FILE,PRG'

```
@EH:FILE,PRG
00, OK,000,000,000,000
```

15.3 Partition management commands

If you want to get the list of partitions, then see section "5 Using partitions"!

15.3.1 Change partition

These commands change the working partition. Partition 0 is not a valid parameter for these commands.

Format:

"CP⟨*partition #*⟩"

"CP"+CHR$(⟨*partition #*⟩)

Examples:

Select partition 2

```
@CP2
02, PARTITION SELECTED,002,000,000,000
```

Select partition 3 the other way

```
open 15,12,15,"cP"+chr$(3):close 15

ready.
```

15.3.2 Get partition info

Get information about a partition. The returned data format is described in Table 27.

Format:

"G-P"[+CHR$(⟨*partition #*⟩)]

Byte	Value	Meaning
0	0	not available
	1	CFS partition
1		Unused
2		Partition number
3–18		~~Partition name~~
19–26		Unused
27–29		Size (65535)

Table 27: G-P data format

```
10 OPEN 15,12,15,"G-P"
20 GET#15,A$,B$,C$:CLOSE 15
30 PRINT"CURRENT PARTITION IS:"ASC(C$+CHR$(0))
```

Listing 41: What's the current partition?

```
10 OPEN 15,12,15,"G-P"+CHR$(2)
20 GET#15,A$,B$,C$:CLOSE 15
30 IF A$="" THEN PRINT"NO SUCH PARTITION!"
```

Listing 42: Is partition 2 there?

15.4 Device management commands

15.4.1 Device number change

If you want to temporary change the device number (until next reset) send '"U0>"+CHR$(⟨*new*⟩)' to the device.

Format:

"U0>"+CHR$(⟨*new drive #*⟩)
"S-8"
"S-9"
"S-D"

Example:

This drive will be device 8 from now on

```
OPEN 15,12,15,"U0>"+CHR$(8):CLOSE15
```

An easier typed variant is the 'S-8', 'S-9' and 'S-D'. The last one restores the default device number.

Examples:

This drive will be device 8 from now on

```
@S-8
00, OK,000,000,000,000
```

Revert to default device number

```
@S-D
00, OK,000,000,000,000
```

15.4.2 Get disk change

The 'G-D' command can be used to get the disk change status byte, which is followed by CHR$(13). If it's non-zero then the disk has been changed or removed since the last operation. Only the next filesystem access will clear the disk change status.

```
10 OPEN 15,13,15,"G-D"
20 GET#15,A$,B$:CLOSE 15:A=ASC(A$+CHR$(0))
30 IF A<>0 THEN PRINT"DISK CHANGED"
```

Listing 43: Disk change detection example

15.4.3 Identify drive

Sending 'UI' or 'U9' on command channel returns the DOS version of the drive.

Format:

"UI"
"U9"

Example:

```
@UI
73, IDE DOS V0.91 IDE64,000,000,000,000
```

15.4.4 Reset drive

Sending 'UJ' or 'U:' on command channel reconfigures the drive.

Format:

"UJ"
"U:"

Example:

```
@UJ
73, IDE DOS V0.91 IDE64,000,000,000,000
```

15.4.5 Power management

It's possible to enter or exit power saving mode if the drive supports it.

Format:

"U0>P0"
"U0>P1"
"U0>P"

Examples:

Spin down drive

```
@U0>P0
00, OK,000,000,000,000
```

Spin up drive

```
@U0>P1
00, OK,000,000,000,000
```

Get power management state. Returned is 1 byte followed by CHR$(13).

```
10 OPEN 15,12,15,"U0>"+CHR$(208)
20 GET#15,A$,B$:CLOSE 15:A=ASC(A$+CHR$(0))
30 IF A=0 THEN PRINT"STANDBY"
40 IF A=128 THEN PRINT"IDLE"
50 IF A=255 THEN PRINT"ACTIVE"
```

Listing 44: Get power state example

15.4.6 Eject or load medium

More useful on CD-ROM, DVD, Zip drive and LS-120 than on hard disk. Don't forget to unlock the medium before eject! (See "15.4.7 Lock or unlock medium"!)

Format:

"U0>E0"
"U0>E1"

Examples:

Load medium (CD-ROM and DVD only)

```
CU0>E0
00, OK,000,000,000,000
```

Eject medium

```
CU0>E1
00, OK,000,000,000,000
```

15.4.7 Lock or unlock medium

It's possible to prevent medium removal on CD-ROM, DVD, Zip and LS-120 drives.

Format:

"U0>L0"
"U0>L1"

Examples:

Unlock medium

```
CU0>L0
00, OK,000,000,000,000
```

Lock medium

```
@U0>L1
00, OK,000,000,000,000
```

15.4.8 Reading time from RTC

Sending 'T-RA' will read the current time in PETSCII, 'T-RB' in BCD, while 'T-RD' in decimal.

Format:

"T-RA"

"T-RB"

"T-RD"

Byte	Meaning
0	Day 0–6: 0→Sunday, 1→Monday, ...
1	Year 00–99: 0→2000, ..., 79→2079, 80→1980, ...
2	Month 1–12: 1→January, ...
3	Date 1–31
4	Hour 1–12
5	Minute 0–59
6	Second 0–59
7	AM/PM: 0→AM, else PM

Table 28: T-RB and T-RD data format

Example:

Get the current time in human readable form (day of week, month, date, year, hour, minutes, seconds, AM or PM).

```
@T-RA
SAT. 08/07/04 07:42:48 PM
```

15.4.9 Format disk

Formatting of floppy disks is sometimes necessary to eliminate bad sectors, or just bring the medium into a usable format. Formatting changes the physical format, it does not create a filesystem on disk. Use CFSfdisk[14] to create a filesystem!

NOTE

During formatting the drive is not accessible (this can take more than 30 min with a really bad disk), fortunately it's possible to use other drives meanwhile. But of course HDINIT will break formatting, so try to avoid using it.

Format:

"N=⟨*format code*⟩"

Code	Meaning
720K	Double density disk with 720 kB capacity (LS-120)
1.2M	High density disk with 1200 kB capacity (LS-120)
1.44M	High density disk with 1440 kB capacity (LS-120)
120M	SuperDisk with 120 MiB capacity (LS-120)

Table 29: Disk format codes

Example:

Format a 1.44 MB disk

```
@N=1.44M
00, OK,000,000,000,000
```

[14] CFSfdisk is described in section "4 Preparing a blank disk".

15.4.10 Write protect

Software write protect switch for the entire drive. This is not permanent, but a simple reset won't disable it.

Format:

"W-0"
"W-1"

Example:

Enable write protection.

```
CW-1
00, OK,000,000,000,000
```

15.5 Direct access commands

To learn more about direct access read section "8 Direct access".

15.5.1 Identify

Format:

"B=R"+CHR$(⟨*channel #*⟩)+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)

15.5.2 Buffer read

Format:

"B=R"+CHR$(⟨*channel #*⟩)+CHR$(⟨*head #*⟩)+
CHR$(⟨*cylinder bits 8–15 #*⟩)+CHR$(⟨*cylinder bits 0–7 #*⟩)+
CHR$(⟨*sector #*⟩)
"B=R"+CHR$(⟨*channel #*⟩)+CHR$(⟨64+*LBA bits 24–27 #*⟩)+

CHR$(⟨*LBA bits 16–23 #*⟩)+CHR$(⟨*LBA bits 8–15 #*⟩)+
CHR$(⟨*LBA bits 0–7 #*⟩)

15.5.3 Buffer position

Format:

"B=P"+CHR$(⟨*channel #*⟩)+CHR$(⟨*position bits 0–7 #*⟩)+
CHR$(⟨*position bits 8–15 #*⟩)
"B-P:"; ⟨*channel #*⟩; ⟨*position bits 0–7 #*⟩; ⟨*position bits 8–15 #*⟩

15.5.4 Buffer write

Format:

"B=W"+CHR$(⟨*channel #*⟩)+CHR$(⟨*head #*⟩)+
CHR$(⟨*cylinder bits 8–15 #*⟩)+CHR$(⟨*cylinder bits 0–7 #*⟩)+
CHR$(⟨*sector #*⟩)
"B=W"+CHR$(⟨*channel #*⟩)+CHR$(⟨64+*LBA bits 24–27 #*⟩)+
CHR$(⟨*LBA bits 16–23 #*⟩)+CHR$(⟨*LBA bits 8–15 #*⟩)+
CHR$(⟨*LBA bits 0–7 #*⟩)

15.6 Directory handling commands

To learn more about using directories see section “6 Using directories”.

15.6.1 Change working directory

All directories of the path must be executable.

Format:

"CD[⟨*partition #*⟩]:⟨*path*⟩"
"CD←"
"CD/⟨*path*⟩"

Example:

```
@CD:NEWDIR
00, OK,000,000,000,000
```

15.6.2 Change root directory

All directories of the path must be executable. Also the current working directory is changed to the new root. This command only works on CFS formatted partitions. There's no way to get back to the partition's real root directory, except the HDINIT BASIC command. A simple reset won't disable this.

Of course if you've changed the root directory of partition 1 on the system drive, then the boot file, manager configuration file, plugins, and the DOS wedge shell will be searched by IDEDOS according to the new root directory.

Format:

"CR[⟨*partition #*⟩]:⟨*path*⟩"
"CR←"
"CR/⟨*path*⟩"

Example:

```
@CR:BBSSANDBOX
00, OK,000,000,000,000
```

15.6.3 Make directory

Create a directory. The new directory must be on a writable partition and in a writable directory.

Format:

"MD[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*directory name*⟩"

Example:

Create the directory called 'NEWDIR'

```
@MD:NEWDIR
00, OK,000,000,000,000
```

15.6.4 Remove directory

Remove a directory. The directory must be on a writable partition and in a writable directory. Watch out for hidden files if IDEDOS refuses to remove an "empty" directory.

Format:

"RD[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*directory name*⟩"

Example:

Remove the directory called 'OLDDIR'

```
@RD:OLDDIR
01, FILES SCRATCHED,001,000,000,000
```

15.6.5 Rename directory header

Rename the directory header. The directory must be writable and on a writable partition.

Format:

"R-H[⟨*partition #*⟩][/⟨*path*⟩/]:⟨*header*⟩"

Example:

Change the header of current directory to 'NEW LABEL'

```
@R-H:NEW LABEL
00, OK,000,000,000,000
```

15.7 CD-ROM related commands

15.7.1 Read TOC

For detailed description see section "8 Direct access".

Format:

"B=T"+CHR$(⟨*channel #*⟩)+CHR$(⟨*format #*⟩)+CHR$(⟨*starting track #*⟩)

15.7.2 Read sub-channel information

For detailed description see section "8 Direct access".

Format:

"B=S"+CHR$(⟨*channel #*⟩)+CHR$(⟨*format #*⟩)+CHR$(⟨*starting track #*⟩)+ CHR$(⟨*mode #*⟩)

15.7.3 Audio playback

This command will play a part of audio CD. Start and end position is specified in MSF (Minute 0–99, Second 0–59, Frame 0–74).

NOTE

For some reason audio tracks start always 2 seconds later than specified in TOC, so for example the first audio track begins at 2 seconds.

Format:

"U0>CA"+CHR$(⟨*end frame #*⟩)+CHR$(⟨*end second #*⟩)+
CHR$(⟨*end minute #*⟩)+CHR$(⟨*start frame #*⟩)+
CHR$(⟨*start second #*⟩)+CHR$(⟨*start minute #*⟩)

15.7.4 Fast forward and reverse

This command starts fast forward and reverse from the specified position until the end of disc. The direction and the position format is specified by the mode byte, as described in Table 30.

LBA position It's specified in sectors from the beginning of the disc, first byte is the least significant, and the fourth the most.

MSF position It's the elapsed time from the beginning of the disc, first byte is the frame, then second and minute, while the fourth is reserved (0).

Track position It's specified by the stating track number, which is the 1st byte, the other 3 are reserved.

Format:

"U0>CF"+CHR$(⟨*pos1 #*⟩)+CHR$(⟨*pos2 #*⟩)+CHR$(⟨*pos3 #*⟩)+
CHR$(⟨*pos4 #*⟩)+CHR$(⟨*mode #*⟩)

Example:

Fast forward from track 2:

```
PRINT#15,"U0>CF"CHR$(2)CHR$(0)CHR$(0)CHR$(0)CHR$(128)
```

Bit	Value	Meaning
0–3		Unused
4	0	Forward
	1	Reverse
5		Unused
6–7	0	LBA position
	1	MSF position
	2	Track position
	3	Reserved

Table 30: Bits of fast forward and reverse mode byte

15.7.5 Pause, resume, and stop audio playback

Format:

"U0>CP0"

"U0>CP1"

"U0>CS"

Examples:

Pause playback

```
CU0>CP0
00, OK,000,000,000,000
```

Continue playback

```
CU0>CP1
00, OK,000,000,000,000
```

Stop playback

```
CU0>CS
00, OK,000,000,000,000
```

15.7.6 Volume control

This command lets control the output volume of CD-ROM drive. Output/channel 0 is the left side, while output/channel 1 is the right.

Format:

"U0>CV"+CHR$(⟨#⟩)+CHR$(⟨#⟩)+CHR$(⟨#⟩)+CHR$(⟨#⟩)+
CHR$(⟨#⟩)+CHR$(⟨#⟩)+CHR$(⟨#⟩)+CHR$(⟨#⟩)

Byte	Meaning
0	Output port 0 channel selection (0–15)
1	Output port 0 volume (0–255)
2	Output port 1 channel selection (0–15)
3	Output port 1 volume (0–255)
4	Output port 2 channel selection (0–15)
5	Output port 2 volume (0–255)
6	Output port 3 channel selection (0–15)
7	Output port 3 volume (0–255)

Table 31: Volume control format

```
10 OPEN 15,12,15
20 V$=CHR$(2)+CHR$(128)+CHR$(1)+CHR$(128)
30 V$=V$+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
40 PRINT#15,"U0>CV"V$
50 CLOSE 15
```

Listing 45: Reverse left and right speakers, and -6dB amplification

```
10 OPEN 15,12,15
20 V$=CHR$(3)+CHR$(255)+CHR$(0)+CHR$(0)
30 V$=V$+CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0)
40 PRINT#15,"U0>CV"V$
```

Value	Meaning
0	Output port muted
1	Audio channel 0
2	Audio channel 1
3	Audio channel 0 and 1 mixed
4	Audio channel 2
8	Audio channel 3
15	All mixed

Table 32: Output channel selection

```
50 CLOSE 15
```

Listing 46: Mono output on left speaker only, full volume

15.7.7 Volume settings query

This command returns the current settings from the CD-ROM drive in a structure described in Table 31, plus a `CHR$(13)` at the end.

Format:

"U0>CV"

```
0 print"drive":input dr
10 open 15,dr,15,"u0>cV"
20 get#15,a$
30 if a$<chr$(16) then 60
40 input#15,b$,c$,d$,e$:rem error
50 print a$;b$,c$,d$,e$:goto 90
60 get#15,b$,c$,d$,e$,f$,g$,h$,i$
70 print"left  channel:"asc(b$+chr$(0))
80 print"right channel:"asc(d$+chr$(0))
90 close 15
```

Listing 47: Print current volume setting

15.7.8 Medium type

This command returns the CD-ROM media type in one byte, plus a `CHR$(13)` at the end.

Format:

"U0>CM"

```
0 PRINT"DRIVE":INPUT DR
10 OPEN 15,DR,15,"U0>CM":GET#15,A$,B$:CLOSE 15
15 IF ST<>64 THEN PRINT"NOT SUPPORTED":END
20 T=ASC(A$+CHR$(0)):A=T AND 15:B=INT(T/16)
30 IF B=7 AND A=0 THEN PRINT"NO CD IN DRIVE":END
40 IF B=7 AND A=1 THEN PRINT"TRAY OPEN":END
50 IF B=7 AND A=2 THEN PRINT"FORMAT ERROR":END
60 IF B=1 AND A<9 THEN PRINT"CD-R ";
70 IF B=2 AND A<9 THEN PRINT"CD-E ";
80 IF B=3 AND A=0 THEN PRINT"HD UNKNOWN":END
90 IF B=3 AND A=1 THEN PRINT"HD 120MM":END
100 IF B=3 AND A=5 THEN PRINT"HD 80MM":END
110 IF B>2 OR A>8 THEN PRINT"?":END
120 IF A>0 AND A<5 THEN PRINT"120MM "
130 IF A>4 AND A<9 THEN PRINT"80MM "
140 IF A=0 THEN PRINT"UNKNOWN"
150 IF A=1 OR A=5 THEN PRINT"DATA"
160 IF A=2 OR A=6 THEN PRINT"AUDIO"
170 IF A=3 OR A=7 THEN PRINT"DATA AND AUDIO"
180 IF A=4 OR A=8 THEN PRINT"MULTISESSION"
```

Listing 48: This small program will display the medium type in CD-ROM drive

15.7.9 Drive capabilities

Returns the drive capabilities in 14 bytes with an extra `CHR$(13)` at the end.

Format:

"U0>CC"

```
0 PRINT"DRIVE":INPUT DR
10 OPEN 15,DR,15,"U0>CC":DIM C(14)
20 FOR A=0 TO 14:GET#15,A$:C(A)=ASC(A$+CHR$(0))
25 NEXT:CLOSE 15
30 IF ST<>64 THEN PRINT"NOT SUPPORTED":END
35 PRINT"DRIVE CAPABILITIES:":PRINT
40 IF C(0) AND 1 THEN PRINT"READS CD-R"
50 IF C(0) AND 2 THEN PRINT"READS CD-RW"
60 IF C(0) AND 4 THEN PRINT"READS CD-R METHOD2"
70 IF C(0) AND 8 THEN PRINT"READS DVD-ROM"
80 IF C(0) AND 16 THEN PRINT"READS DVD-R"
90 IF C(0) AND 32 THEN PRINT"READS DVD-RAM"
100 IF C(1) AND 1 THEN PRINT"WRITES CD-R"
110 IF C(1) AND 2 THEN PRINT"WRITES CD-RW"
120 IF C(1) AND 4 THEN PRINT"WRITES SIMULATION"
130 IF C(1) AND 16 THEN PRINT"WRITES DVD-R"
140 IF C(1) AND 32 THEN PRINT"WRITES DVD-RAM"
150 IF C(2) AND 1 THEN PRINT"PLAYS AUDIO"
160 IF C(2) AND 2 THEN PRINT"AUDIO/VIDEO STREAM"
170 IF C(2) AND 4 THEN PRINT"DIGITAL OUT PORT1"
180 IF C(2) AND 8 THEN PRINT"DIGITAL OUT PORT2"
190 IF C(2) AND 16 THEN PRINT"READS MODE2 FORM1"
200 IF C(2) AND 32 THEN PRINT"READS MODE2 FORM2"
210 IF C(2) AND 64 THEN PRINT"READS MULTISESSION"
220 IF C(3) AND 1 THEN PRINT"READS CDDA"
230 IF C(3) AND 2 THEN PRINT"READS CDDA CONTINUE"
240 IF C(3) AND 4 THEN PRINT"RW SUPPORTED"
250 IF C(3) AND 8 THEN PRINT"RW CORRECTION"
260 IF C(3) AND 16 THEN PRINT"C2 POINTER SUPPORT"
270 IF C(3) AND 32 THEN PRINT"ISRC CODE SUPPORT"
```

```
280 IF C(3) AND 64 THEN PRINT"UPC CODE SUPPORT"
290 IF C(4) AND 1 THEN PRINT"MEDIA LOCKABLE"
300 PRINT"MEDIA CURRENTLY ";
302 IF (C(4) AND 2)=0 THEN PRINT"UN";
305 PRINT"LOCKED"
310 PRINT"MEDIA LOCK JUMPER ";
312 IF (C(4) AND 4)=0 THEN PRINT"NOT ";
315 PRINT"SET"
320 IF C(4) AND 8 THEN PRINT"MEDIA EJECTABLE"
330 PRINT"LOADING MECHANISM: ";:A=INT(C(4)/32)
340 IF A=0 THEN PRINT"CADDY"
350 IF A=1 THEN PRINT"TRAY"
360 IF A=2 THEN PRINT"POP-UP"
370 IF A=3 OR A>5 THEN PRINT"UNKNOWN"
380 IF A=4 THEN PRINT"CHANGER WITH DISCS"
390 IF A=5 THEN PRINT"CHANGER USING CARTRIDGE"
400 IF (C(5) AND 1)=0 THEN PRINT"NO ";
405 PRINT"SEPARATE VOLUME CONTROL"
410 IF (C(5) AND 2)=0 THEN PRINT"NO ";
420 PRINT"SEPARATE VOLUME MUTING"
430 IF C(5) AND 4 THEN PRINT"CAN REPORT SLOT"
440 IF C(5) AND 8 THEN PRINT"CAN SELECT SLOT"
450 SP=INT((C(6)*256+C(7)+88)/176)
460 PRINT"MAXIMUM SPEED:"SP"X"
470 PRINT"VOLUME LEVELS:"C(8)*256+C(9)
480 PRINT"BUFFER SIZE:"C(10)*256+C(11)"KB"
490 PRINT"CURRENT SPEED:";
500 PRINT INT((C(12)*256+C(13)+88)/176)"X"
```

Listing 49: This little longer program will display all information this command can return.

15.8 Misc commands

15.8.1 Memory read

It's included for compatibility. It reads from a fake ROM filled with the message 'IDE64 CARTRIDGE ' or depending on the 'CMD emulation' setting[15] in the setup utility it can also be the text 'CMD HD EMULATED IDE64 CARTRIDGE '.

Format:

"M-R"+CHR$(⟨*address bits 0–7 #*⟩)+CHR$(⟨*address bits 8–15 #*⟩)+
CHR$(⟨*number of bytes #*⟩)

15.8.2 Memory write

It's included for compatibility, write to $77–$78 changes device number, otherwise it has no effect.

Format:

"M-W"+CHR$(⟨*address bits 0–7 #*⟩)+CHR$(⟨*address bits 8–15 #*⟩)+
CHR$(⟨*number of bytes #*⟩)+CHR$(⟨*first byte #*⟩)+...

15.8.3 Validate

It's included for compatibility, has no effect. Use the CFSfsck[16] utility to check filesystem integrity.

Format:

"V[⟨*partition #*⟩]"

[15] See section "3.1.11 CMD emulation".
[16] CFSfsck is described in section "19 Filesystem checking".

16 IDEDOS error messages

These are the possible returned error codes on channel #15 with their short descriptions.

00: OK *(not an error)*
The last action finished without errors.

01: FILES SCRATCHED *(not an error)*
This message appears after removing files or directories. The first two numbers represent the number of files removed in little endian order.

02: PARTITION SELECTED *(not an error)*
The partition selection was successful, the selected partition is the first number.

NOTE

All message codes below 20 can be ignored, real errors have numbers of 20 or more.

20: READ ERROR *(unrecoverable error)*
Unrecoverable error, probably there's a bad sector on medium. The numbers represent the starting sector where the error happened.

21: READ ERROR *(timeout during read)*
Timeout happened while reading data from drive. Might be a hardware problem. The numbers represent the starting sector where the error happened.

22: READ ERROR *(unformatted medium)*
The inserted medium needs to be formatted physically. This message can also happen for a blank optical medium. The numbers represent the starting sector where the error happened.

23: READ ERROR *(medium error)*
Cannot read data due to bad block, read was aborted by drive. The numbers represent the starting sector where the error happened.

25: WRITE ERROR *(verify error)*
Cannot write data due to bad block, or the written data does not match. Write aborted. The numbers represent the starting sector where the error happened.

26: WRITE PROTECT ON
The file, directory, partition or device is write protected, or the drive reports a data protect error during write, or tried to write to a write protected medium.

27: ACCESS DENIED
Cannot read the read the content of a non-readable directory, or the drive reports data protect error during read. The numbers represent the starting sector where the error happened.

28: WRITE ERROR *(timeout during write)*
Timeout happened while writing data to drive. Might be a hardware problem. The numbers represent the starting sector where the error happened.

29: DISK CHANGED
Disk change detected, all open files lost on the device.

30: SYNTAX ERROR *(general syntax)*
Syntax error in command, something is missing.

31: UNKNOWN COMMAND
Unknown or unimplemented command, or this command is not supported by the drive.

32: SYNTAX ERROR *(long line)*
Command buffer overflow, the sent command string is too long to fit in the buffer.

33: SYNTAX ERROR *(invalid file name)*
Illegal character in file name, or filetype, or unknown short filetype. Do not use wildcards in the file name when creating files.

34: SYNTAX ERROR *(missing file name)*
Missing file name for command. Probably a missing colon.

39: PATH NOT FOUND
A part of path was not found, or tried to move file to different partition using rename. Possible link loop found, or buffer overflow during link expansion. Can also happen when accessing files with slash in the file name, but without using CMD emulation.

40: TIMEOUT ERROR
The PCLink server did not answer in time, or it's not present.

41: FRAME ERROR
Telegram format error detected in PCLink transaction.

42: CRC ERROR
Checksum error detected in PCLink telegram.

50: RECORD NOT PRESENT
The record read is beyond the end of file, or tried to seek beyond the end of file. This is not really an error, but it signals that the file will be expanded.

51: OVERFLOW IN RECORD
The written record got truncated, or tried to seek beyond the end of record.

52: FILE TOO LARGE
Tried to seek beyond the maximum file size, or tried to write too much to a file.

60: FILE OPEN ERROR
Better close opened files, before remove or move. Also do not try to remove the working directory. Multiple open of an already open file is not allowed, expect all opens are read only.

62: FILE NOT FOUND
The named file was not found.

63: FILE EXISTS
Cannot create file or directory with this name, or cannot rename file to this name. Remove the existing file first, or use replace.

64: FILE TYPE MISMATCH
Cannot change this file type to the specified with rename, or cannot create file with this type.

66: ILLEGAL REQUEST
Illegal request during read, tried to access beyond end of disk, track 0 not found, or no such sector.

67: ILLEGAL REQUEST
Illegal request during write, tried to access beyond end of disk, track 0 not found, or no such sector.

70: NO CHANNEL
Channel number incorrect for direct access commands, or for the seek command. Tried to reopen an already open channel, or no more free buffers left. Invalid direct access channel, directory read channel or relative file channel selected.

71: DIR ERROR
Directory header not found, filesystem might be damaged.

72: PARTITION FULL
There's no more space left on partition, or tried to create more than 1023 entries in a directory.

73: IDE DOS V0.91 IDE64 *(ide64, cdrom, fdd64, pclink)*
Displayed after reset, it's the DOS version, and drive type. Might also happen if operation is unsupported on a non-native filesystem.

74: DRIVE NOT READY
Drive not ready for command, there's no disk in drive, or command aborted by drive. Can also happen on unexpected response from drive, or when PCLink communication is not possible.

75: FORMAT ERROR
Unknown filesystem on medium.

76: HARDWARE ERROR
The drive reports hardware error, or cabling problem, unreliable communication with drive. Try to use shorter IDE-cable or ShortBus cable and connect the cartridge directly into the computer.

77: SELECTED PARTITION ILLEGAL
The selected partition does not exists, or the filesystem is unsupported. The the first number is the illegal partition number.

17 Compatibility

17.1 Hardware

17.1.1 Commodore serial drives, datassette

Of course they work. There's also fast load and save support for 1541, 1570, 1571 and 1581. Datassette is supported, if KERNAL supports it.

17.1.2 SuperCPU accelerator

The SuperCPU IDEDOS expects emulation mode with direct page starting at the 0th byte of bank 0, like normal. The high byte of register A is not preserved during operation. Tested on SCPU64V2 with SuperCPU DOS 2.04.

17.1.3 Retro Replay, Action Replay, Final Cartridge

These hardware use the same I/O space as the IDE64 cartridge, so they won't work.

17.1.4 RR-Net ethernet card, SilverSurfer serial port

Works fine when attached to the clock-port of the V4.1 or V4.2 version of the cartridge. With V2.1 it works if Retro Replay is jumpered to be flashed. (this way there's no I/O space conflict) Versions V3.1, V3.2, V3.4 and V3.4+ require a cartridge port expander, otherwise it's the same as for V2.1.

17.1.5 MMC64 cartridge

No conflict, but needs support in IDEDOS. Also there could be timing problems resulting in unreliable operation.

17.1.6 CMD RamLink RAM expansion

Probably does not work on C64, but needs more testing. RL-DOS won't work with IDEDOS if using SuperCPU.

17.1.7 CMD HD series, CMD FD series

As these are serial drives and they should work. They are supported by the manager too.

17.1.8 64HDD hard disk emulator

With some versions of the 64HDD hard disk emulator program you have to turn off the "3.1.3 Floppy speeder" option in the setup. (The problem is that 64HDD simply can't deal with the multiple channels open at the same time on the drive case...)

17.1.9 Commodore 128

Works in C64 mode, clears `$D02F`–`$D030`. The cartridge is not designed for 2 MHz operation, so do not call IDE64 routines in 2 MHz mode! C128 keyboard is available if cartridge has cartconfig register and support is compiled in. (NUMERIC KEYPAD, CURSOR KEYS, ESC, TAB, LINEFEED are available, HELP calls monitor, ALT and NOSCROLL unused)

17.1.10 JiffyDOS speed enhancement system

It's detected and used. (tested with v6.01) The built-in DOS Wedge has higher priority, if you want to use the JiffyDOS one then disable it in the setup utility!

Loading, saving and file reading in manager is accelerated if the drive has JiffyDOS, even if there's no JiffyDOS ROM installed in the computer (selectable at compile time). If you get sometimes a '?LOAD ERROR' during the load of directory from a JiffyDOS drive, then that's not a bug in IDEDOS. The original JiffyDOS load routine tries to workaround this by retrying IECIN, and this causes an endless loop until STOP is pressed. I prefer getting an error over waiting forever...

17.1.11 Dolphin DOS

Loading, saving and copying in manager is accelerated if the drive has Dolphin DOS and parallel cable, even if there's no Dolphin DOS ROM installed in the computer (selectable at compile time).

17.1.12 NTSC/PAL systems

Both works, there should be no timing issues.

17.1.13 RAM Expansion Unit

Works, not touched. RamDOS does not work, but if someone is interested it's possible to support it.

17.1.14 Second SID

Supported, base location has to be selected on compile. The default is `$D420`. (it's muted on STOP + RESTORE, and the manager can use it for warning sound)

17.1.15 +60K memory expansion

Works, not touched.

17.1.16 CBM IEEE-488 interface

IEEE-488 does not work, because the IDE64 cartridge cannot manipulate the EXROM line.

17.1.17 SwiftLink-232 ACIA cartridge

The registers of SwiftLink-232 are mirrored in the whole I/O area, but after some modifications it works. Supported for serial PCLink.

17.1.18 Turbo232 UART cartridge

Works fine, may need an cartridge port expander. Supported for serial PCLink by IDEDOS.

17.1.19 CHS or LBA hard disk

Use modern LBA capable hard disks and LBA formatted partitions if possible. Then it's much faster to scratch files, check filesystem, etc.

because there's no CHS→LBA→CHS translation. This translation is slow due to multiplications and divisions required for the conversion.

17.2 Software

Auto starting programs are not supported when started from IDEDOS devices and the built-in floppy speeder is disabled when loading them from floppy. Also these type of programs disable IDEDOS most of the time by overwriting the vector table during load.

Programs using serial bus specific routines, custom loaders (IRQ or fast loaders), and direct disk access won't work with IDEDOS devices of course, as the IDE64 cartridge is neither attached to the serial bus, nor is a floppy drive emulator.

18 Updating IDEDOS

The IDE64 interface cartridge is equipped with 64, 128 or 512 KiB flash memory (AT29C512, AT29C010, SST39SF040) for the firmware. The content of flash memory is non-volatile and can be updated, so you can always run the latest firmware with the new features and bug fixes.

Updating IDEDOS is very easy and does not require any special skills. The cartridge was designed so that it's always possible to update the firmware even if it was messed up, so you can't render your cartridge unusable by an interrupted update or wrong firmware.

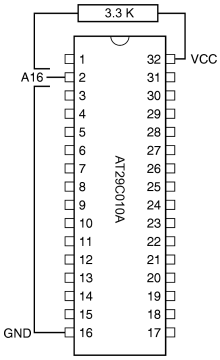


Figure 8: 128 KiB PEROM upgrade for cartridges older than V3.4+

With a SuperCPU equipped machine it's necessary to change between the two versions of IDEDOS depending on the presence of SuperCPU. Updating firmware every time you want to use IDE64 with or without the SuperCPU can be frustrating. By using a 128 or 512 KiB flash memory it's possible to change the firmware fast and avoid time wasting updates. (starting from version V3.4+ of the cartridge, upgrade of older cartridges is possible but requires soldering and electronic skills!)

Here's a short guide to updating IDEDOS:

1. Get the latest Perom programmer and the new firmware and copy both to a serial bus drive. (e.g. floppy drive, CMD HD,

etc.) If you've got a PCLink cable it's also possible to use it for update. (and it's much faster)

2. Plug the cartridge into the expansion port first, while the computer is switched off. Removing of SuperCPU is not necessary, it can speed up the update (JiffyDOS and 20 MHz), however if you are having trouble you may remove it. The IDE64 V4.1 and V4.2 cartridge can only be updated when the SuperCPU is disabled.

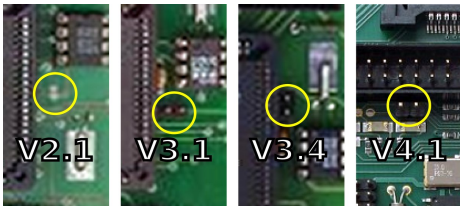


Figure 9: Location of the PGM pin on different cards

3. There are two squares near the big Lattice chip and the 8 pin DS1302 called PGM. By connecting these 2 squares on board together you can enable flash update. (use screwdriver) Newer version of the IDE64 cartridge have pins instead of squares, and you have to use a jumper to connect them. (see picture) The V4.1 and V4.2 cartridge has the programming jumper near the IDE port.

4. If using a greater than 64 KiB flash memory, then select the bank you want to program by the switch!

WARNING!

Never change bank or set the programming pins while IDEDOS is running! You will trash your disks seriously.

5. Now you can turn on your computer or press reset if it's already on. The green led should blink now, except if you have your SuperCPU enabled, then it's off. If the led is steady on it means the flash write protect is still enabled, this case retry from the beginning. (it's not easy to connect the 2 squares on older cartridges for the first try...)

6. The normal C64 screen should appear, the only difference is that there are now only '30719 BYTES FREE'. Now you can remove the screwdriver or jumper.

7. Load the Perom programmer and start it. It should start with the screen shown in the picture. (if using v1.1) Now you may backup your current IDEDOS if you want, and update to the latest from PCLink or disk. At least version v1.5 is required for the V4.1, version v1.9 for the V4.2 cartridge. Now select the action with the CRSR keys and hit RETURN.

NOTE

Erasing of PEROM chip is not needed before programming, so do not select it unless you have reasons to do so. It cleans out all 64 KiB banks!

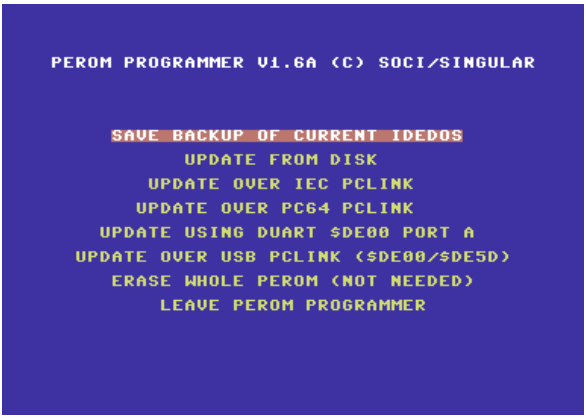

PEROM PROGRAMMER V1.6A (C) SOCI/SINGULAR
SAVE BACKUP OF CURRENT IDEDOS
UPDATE FROM DISK
UPDATE OVER IEC PCLINK
UPDATE OVER PC64 PCLINK
UPDATE USING DUART $DE00 PORT A
UPDATE OVER USB PCLINK ($DE00/$DE5D)
ERASE WHOLE PEROM (NOT NEEDED)
LEAVE PEROM PROGRAMMER

Figure 10: The Perom programmer utility

8. After selecting one of the update menu items the program will ask for device number and filename. Leaving the filename blank cancels the update. If the program says it cannot find the PEROM then make sure you removed the jumper! (also it's not recommended using pencil for connecting the squares because the remaining carbon may permanently connect them unless cleaned)

9. If the filename was correct and the PCLink server was running then the programming should start. A box filled with characters will appear on screen each character representing a 128 byte sector of flash memory. During the rewrite this area will be filled. If finished you get an "Update successful" message. If hexadecimal numbers appear in the right corner of the

screen and you get “Update failed” then better clean the contacts of your cartridge and connect it directly without any port expander. This can also mean that you’ve exceeded the typical 10000 rewrites for some sectors, which is very unlikely. (or the flash memory is just failing for some other unknown reason) Also disk errors may interrupt the update, this case get a new disk and retry the update.

10. Now if ready (“Update successful”) you may leave the program and press reset to start the new firmware. If using older versions of the firmware (before 0.9x) then power cycle the computer!

19 Filesystem checking

As you might already noticed there's no built in "validate" command in IDEDOS. This is not a mistake, checking and repairing the complex structures of the CFS filesystem as fast as possible needs lots of memory, and sometimes choices have to be made.

The filesystem integrity check tool for the CFS filesystem is called "CFS 0.11 Filesystem check v0.05a", or short "CFSfsck".

The partition table integrity is checked by the CFSfdisk tool. Each partition holds a separate CFS filesystems, this is what's checked by CFSfsck.

This tool can check one filesystem (partition) at a time by looking into all directories and checking all metadata for problems. It will automatically recover space not allocated to any file (not likely, but non-closed files may sometimes cause this), and allocates non-allocated file data if any (this shouldn't happen). Also it'll remove all non-closed files. Everything else needs a confirmation.

19.1 Using CFSfsck

After start the tool it will ask for the device, type of drive, and partition. Only the LS-120 and Zip drives are floppy drives.

Drive numbers: 0 = primary master, 1 = primary slave, 2 = secondary master, 3 = secondary slave.

The process can take a few minutes, 100 MiB is done in $\sim$6 min, but this is may vary with the configuration. Using modern LBA capable disks and enabling of the SuperCPU will help a lot. For a minimal speed up hold SHIFTLOCK down. (only the RIGHTSHIFT + C= combo can be used to toggle the character set)

There can be various error messages with questions. Turn up the volume, and you'll notice the alarm sound when there's any need for further user action.

After CFSfsck finishes various filesystem statistics like disk space used, number of files and directories gets displayed as a bonus. Also during the process you can watch the upper part of the screen for progress indication.

19.2 Errors and resolutions

Here's a list of possible errors during the filesystem check:

Invalid root directory The root directory is not inside the partition, or does not have a valid signature, or could not be read because of a disk error.
Choices: none
Fix: Manually by disk editor, or reformatting of the partition.

Invalid filename Some special characters reserved for internal use (e.g. wildcards) were found in filename or filetype.
Choices: `Ignore`: do nothing, `Fix`: replace them with space

Directory structure too deep Directory nesting too deep, CFSfsck has run out of memory.
Choices: none
Fix: Move deep directories around manually, and rerun CFSfsck.

Invalid directory The directory does not have a valid signature, or could not be read because of a disk error.

Choices: `Ignore`: Do nothing, `Remove`: Delete the directory, `Abort`: exit

Filelength too short There's additional data beyond the file's end, which cannot be accessed, because the file's size in the directory entry is incorrect. Most likely the file was not closed after an append, or relative file expansion.
Choices: `Ignore`: Do nothing, `Fix`: Increase the file's length.
Note: If the current size is OK, then make a copy of this file and delete the original. This may be the case with some old beta IDEDOS 0.9x versions.

Crosslinked file Blocks belonging to this file are already allocated in another file.
Choices: `Ignore`: Do nothing, `Remove`: Delete the file, `Abort`: exit
Note: It's not sure that actually this file is damaged, maybe the other one is overwritten by this one. Check this file manually, and if OK, do a copy before removing.

Invalid sector address A sector belonging to a file is not in partition bounds or could not be read because of a disk error.
Choices: `Ignore`: Do nothing, `Remove`: Delete the file, `Abort`: exit
Note: You may recover parts of the file by trying to copy it.

Crosslinked directory Directory blocks already allocated in another file.
Choices: `Ignore`: Do nothing, `Truncate`: Truncate the directory list, `Abort`: exit

Invalid directory sector A block belonging to the directory is not in partition bounds or could not be read because of a disk error.
Choices: `Truncate`: Truncate the directory list, `Abort`: exit

Write error Disk error during write, probably a bad sector
Choices: `Retry`: Retry, `Abort`: exit

Read error Disk error during read, probably a bad sector
Choices: `Retry`: Retry, `Ignore`: Sure, I know it's bad, do something, `Abort`: exit

CFS disklabel not found Disk is unformatted, or wrong version of filesystem.
Fix: `Fix`: Write new ident, `Abort`: exit
Note: Make sure that this version of CFSfsck is the right one for this version of IDEDOS.

Unknown filetype The filetype is not regular, relative, directory or link.
Choices: `Ignore`: Do nothing, `Remove`: Delete the file, `Abort`: exit
Note: Make sure that this version of CFSfsck is the right one for this version of IDEDOS.

20 Frequently Asked Questions

I copied some programs from floppy to IDE64, but some of them got shorter by a few blocks. Is this a bug?

No, it's just a difference of block size. Traditional CBM and CMD equipment has a block size of 254 bytes, while IDE64 drives have a virtual 256 byte block size (in reality it's 512 or 2048 bytes depending on the medium used). A program which is 49920 bytes long will be 197 block long on floppy and 195 on IDE64.

Is it true that IDE64 can only be used to store one filer games?

Unfortunately most multi part games are written too 1541 or serial bus specific. If you want your favourite game fixed for IDE64, then ask someone who is able to do this. Looking at the IDE64 warezsite will give you some hints about these persons or groups. ;-) And no, you can use IDE64 for much more!

Will IDE64 read/write my DOS formatted floppy with LS-120, or my CompactFlash card from my camera?

Yes, IDEDOS can read FAT12/16/32 filesystems up to 128 GiB with or without partition table up to 8 partitions per drive. Cluster sizes of power of two from 0.5–64 KiB will work. Only short filenames are supported. It's possible to create a hybrid disk partitioning with both FAT and CFS partitions. Due to memory limitations there won't be direct write support included in IDEDOS, this must be coded as an external applica-

tion. And no, NTFS won't be ever supported. (unless you code it)

How comes all the stuff to an IDEDOS filesystem?

The easiest way is to get a CD-ROM and burn all your stuff to CD, and then use the builtin file manager to copy files. Also you can use Star Commander or similar utility with a floppy drive. You can use a virtual serial bus drive emulation program like 64HDD too. Or build or buy a PCLink cable and use that for the transfer. Also it's possible to use Contiki or Wings with an Ethernet cartridge and download the stuff from the Internet. The fastest method is to use the CFS 0.11 FUSE module for mounting and filling the filesystem. Also you can use VICE on Linux to transfer to files when the emulated disk is the block device with the correct geometry. Or use a whole-disk imaging program on windows and configure the image for VICE.

Ok, now I want to backup my IDE64 drive. What are the possibilities?

As the file manager supports recursive copying it's only a matter of selecting all the directories you want, and then copy it to another drive. (like another HDD, ZIP disk, LS-120 disk, CompactFlash card, PCLink, CMD drive, floppy, 64HDD, etc.) Beware of limited filename and directory support of non-IDE64 drives! Also you can mount CFS disks on Linux and other systems using the CFS 0.11 filesystem driver for FUSE. Alternatively it's possible use 'dd' to create an image on POSIX systems, or any whole-disk imaging backup program.

21 GNU Free Documentation License

Version 1.3, 3 November 2008
Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed

as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed

for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply

to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means,

this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

A The ShortBus

This section is about the ShortBus connector of the IDE64 cartridge. It's a 34 pin connector containing a selection of processor and some extra decoded signals. It was designed for connecting extra hardware to the IDE64 cartridge.

On IDE64 V4.1 the jumper JP_2 located next to the ShortBus connector can be used to swap the meaning of $CSEL_0$ and $CSEL_1$ signals when shorted. This swaps the address range `$DE00`–`$DE0F` with `$DE10`–`$DE1F`.

On IDE64 V4.2 the $CSEL_2$ and $CSEL_3$ signals are not available as they were reused for addressing.

WARNING!

Although it's looking similar to a pc floppy connector but *it is not a floppy controller interface*, so never connect a floppy drive or other non-ShortBus hardware, or you'll damage your C64 or IDE64 card!

Here's a short description of ShortBus peripherals I'm aware of.

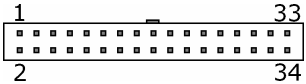


Figure 11: ShortBus female cable connector

ETH64 – Ethernet card

A LAN91C96 chip based Ethernet card. Chip features: Full duplex switched Ethernet support; Enhanced transmit queue management; 6 KiB of on-chip RAM; Supports IEEE 802.3 (ANSI

8802-3) Ethernet standards; Automatic detection of TX/RX polarity reversal; Enhanced power management features; Receive counter for enhanced early receive; Packet memory management unit; Automatic retransmission, bad packet filtering, and transmit padding; External and internal loopback modes; Four direct driven LED outputs for status and diagnostics.

There are 2 jumpers on board: JP_1 for selecting address space `$DE00`–`$DE0F` (1-2, $CSEL_0$) and `$DE10`–`$DE1F` (2-3, $CSEL_1$), and JP_2 for enabling NMI generation. It's supported by Contiki, Wings and some other software.

For more information visit http://www.ide64.org/
Chip datasheet can be found at http://www.smsc.com/

DUART – dual port RS-232 interface

A XR68C681 based dual RS-232 card featuring: Two full duplex, independent channels; Asynchronous receiver and transmitter; Dual buffered transmitter, quadruple-buffered receiver; Programmable stop bits in 1/16 bit increments; Internal baud rate generators with 23 different baud rates from 50 to 115200; Independent baud rate selection for each transmitter and receiver; Normal, autoecho, local loopback and remote loopback modes; Multi-function 16 bit counter or timer; Interrupt output with eight maskable interrupt conditions; Interrupt vector output on acknowledge; 8 general purpose outputs; 6 general purpose inputs with change of states detectors on inputs; Standby mode to reduce operating power.

There are 2 jumpers on board: JP_1 for selecting address space `$DE00`–`$DE0F` (1-2, $CSEL_0$) and `$DE10`–`$DE1F` (2-3, $CSEL_1$),

and JP_2 for enabling NMI generation. It's supported by Contiki, Wings, Novaterm 9.6 and some other software.

For more information visit http://www.ide64.org/
Chip datasheet can be found at http://www.exar.com/

DigiMAX – 4 channel 8 bit DAC

A MAX506 based 4 channel 8 bit digital to analog converter card. Simple programming interface, 4 registers represent the four outputs, the written byte will appear as a voltage level between 0 V and 5 V. The output comes out on 2 jack plugs. This card is supported by Modplay, Wings and maybe some other programs. The base address is selectable by jumper: IO1 `$DE40`–`$DE47` or IO2 `$DE48`–`$DE57`.

For more information visit
http://www.jbrain.com/vicug/gallery/digimax/
Chip datasheet can be found at http://www.maxim-ic.com/

ETFE – Ethernet card

CS8900 based Ethernet card, featuring: Full duplex operation; 4 KiB RAM buffer for transmit and receive frames; Automatic polarity detection and correction; Automatic re-transmission on collision; Automatic padding and CRC generation; Automatic rejection of erroneous packets; LED drivers for link status and LAN activity; Standby and suspend sleep modes.

The Ethernet card has one jumper only, which enables chip reset. The jumpers on the ShortBus interface: JP_7 selects address space `$DExx` (1-2), `$DFxx` (2-3). If JP_7 is set to 1-2, then JP_5 selects `$DE00`–`$DE0F` (1-2, $CSEL_0$), `$DE10`–`$DE1F` (2-3, $CSEL_1$).

If the address space is set to `$DFxx`, then JP_8 selects the exact memory location. The card works fine with Contiki and software supporting the original TFE card.

For more information visit http://c64.rulez.org/etfe/
Chip datasheet can be found at http://www.cirrus.com/

Pin	Name	Description
1	GND	Ground
2	V_{CC}	+5 V
3	$\overline{\text{RESET}}$	Reset signal, active low
4	CSEL_4	Chip select signal, active high $DE58–$DE59
5	R/$\overline{\text{W}}$	Read/Write signal from processor
6	$\overline{\text{CSEL}_3}$	Chip select signal, active low $DE48–$DE57
7	ϕ_2	Phi2 clock signal from processor
8	$\overline{\text{CSEL}_2}$	Chip select signal, active low $DE38–$DE47
9	BA	Bus Available, control signal from VIC II
10	$\overline{\text{CSEL}_1}$	Chip select signal, active low $DE10–$DE1F
11	DOT clock	Clock signal from VIC II
12	$\overline{\text{CSEL}_0}$	Chip select signal, active low $DE00–$DE0F
13	$\overline{\text{NMI}}$	Non-Maskable Interrupt, active low
14	$\overline{\text{I/O}_2}$	Chip select signal, active low
15	$\overline{\text{IRQ}}$	Interrupt Request, active low
16	$\overline{\text{I/O}_1}$	Chip select signal, active low
17	D_7	Data bus bit 7
18	A_7	Address bus bit 7
19	D_6	Data bus bit 6
20	A_6	Address bus bit 6
21	D_5	Data bus bit 5
22	A_5	Address bus bit 5
23	D_4	Data bus bit 4
24	A_4	Address bus bit 4
25	D_3	Data bus bit 3
26	A_3	Address bus bit 3

27	D_2	Data bus bit 2
28	A_2	Address bus bit 2
29	D_1	Data bus bit 1
30	A_1	Address bus bit 1
31	D_0	Data bus bit 0
32	A_0	Address bus bit 0
33	V_{CC}	+5 V
34	GND	Ground

Table 33: ShortBus pinout

B The clock-port

This section is about the clock-port connector of the IDE64 V4.1 and V4.2 cartridge. It's a 22 pin connector containing a selection of processor and some extra decoded signals. It can be used to connect extra hardware to IDE64. Unlike other cartridges all 16 registers are accessible for a better compatibility with Amiga accessories.

The JP_2 jumper located near to the ShortBus connector is for swapping the address range `$DE00`–`$DE0F` with `$DE10`–`$DE1F`. This jumper affects the ShortBus addresses too.

WARNING!

When using ShortBus cards and clock-port devices at the same time, make sure that the ShortBus device is configured to *not use the $CSEL_0$ signal*, otherwise the address collision could damage the cards, C64 or IDE64 card!

Here's a short description of clock-port peripherals I've heard of.

ETH64 II – Ethernet card

A LAN91C96 chip based Ethernet card. This chip features: Full duplex operation; Supports enhanced transmit queue management; 6 KiB of on-chip RAM; Supports IEEE 802.3 (ANSI 8802-3) Ethernet standards; Automatic detection of TX/RX polarity reversal; Enhanced power management features; SimulTasking early transmit and early receive functions; Enhanced early transmit function; Receive counter for enhanced early receive; Hardware memory management unit; Automatic retransmission, bad packet rejection, and transmit padding; External

and internal loopback modes; Four direct driven LED outputs for status and diagnostics.

There's one jumper on board for enabling NMI generation. The card is supported by Contiki, Wings and some other software.

The ETH64 II needs all 16 registers of the clock-port for correct operation, which makes it incompatible with cartridges which do not provide them all. (e.g. Retro Replay)

For more information visit http://www.ide64.org/eth64v2.html
Chip datasheet can be found at http://www.smsc.com/

RR-Net – Ethernet card

CS8900 based Ethernet card, featuring: Full duplex operation; 4 KiB RAM buffer for transmit and receive frames; Automatic polarity detection and correction; Automatic re-transmission on collision; Automatic padding and CRC generation; Automatic rejection of erroneous packets; Boundary scan and loopback test; Link status and LAN activity LEDs; Standby and suspend sleep modes.

The card works fine with Contiki and software supporting the RR-Net card.

For more information visit
http://www.schoenfeld.de/inside/Inside_RRnet.txt
Chip datasheet can be found at http://www.cirrus.com/

SilverSurfer – RS-232 interface

16C550 based RS-232 card, featuring: Asynchronous receiver and transmitter; Full duplex operation; 16 byte transmit and receive FIFO; Baud rate generator for rates from 50 to 460800.

The card works fine with Novaterm.

For more information visit
http://rr.c64.org/silversurfer/docs/Inside_RetroSurfer.txt

MP3@64 – MPEG 1/2 Layer 2/3 Audio decoder

MAS3507 based MPEG decoder card, featuring: MPEG 1/2 layer 2 and 3 decoder; support for MPEG 2.5 low bit rates; Variable bit rate support; Up to 320 kbit and 48 kHz.

The card can be used to play MP3 files with a Manager plugin.

For more information visit
http://www.schoenfeld.de/inside/Inside_MP3AT64.txt

Pin	Name	Description
1	GND	Ground
2	V_{CC}	+5 V
3	$\overline{NMI}$	Non-Maskable Interrupt, active low
4	$\overline{CSEL_0}$	Chip select signal, active low `$DE00`–`$DE0F`
5	NC	Not connected
6	NC	Not connected
7	$\overline{IORD}$	Read signal, active low
8	$\overline{IOWR}$	Write signal, active low
9	A_3	Address bus bit 3
10	A_2	Address bus bit 2
11	A_1	Address bus bit 1
12	A_0	Address bus bit 0
13	D_7	Data bus bit 7
14	D_6	Data bus bit 6
15	D_5	Data bus bit 5
16	D_4	Data bus bit 4
17	D_3	Data bus bit 3
18	D_2	Data bus bit 2
19	D_1	Data bus bit 1
20	D_0	Data bus bit 0
21	GND	Ground
22	$\overline{RESET}$	Reset signal, active low

Table 34: Clock-port pinout

C More information

Online resources about the IDE64 cartridge and related material. It's just a short collection, so you may also use your searching skills to get more. ;-)

C.1 Related Internet sites

The IDE64 project homepage http://ide64.org/
The IDE64 project's homepage with information about the cartridge, peripherals, and lot more.

The IDE64 Information Portal http://news.ide64.org/
The latest news concerning IDE64.

The IDE64 warez site http://warez.ide64.org/
Lots of stuff to fill your empty disks.

The IDEDOS project page http://idedos.ide64.org/
The latest version of IDEDOS can be found here.

The IDE64 list http://groups.google.com/group/ide64/
Subscribe to the list and get your questions answered.

C.2 Distributors

The IDE64 card is currently not available through distributors, please ask Josef directly at soucek.josef@gmail.com for pricing and availability.

D Acronyms

ACIA	Asynchronous Communications Interface Adapter
ATA	AT Attachment
ATAPI	ATA Packet Interface
ATIP	Absolute Time In Pre-groove
ATX	Advanced Technology Extended
BCD	Binary Coded Decimal
CBM	Commodore Business Machines
CD	Compact Disc
CD-ROM	Compact Disc Read Only Memory
CF	CompactFlash
CFS	Commodore File System
CHS	Cylinder Head Sector
CIA	Complex Interface Adapter
CMD	Creative Micro Designs
CRC	Cyclic Redundancy Check
DOS	Disk Operating System
DVD	Digital Versatile Disc
DUART	Dual Universal Asynchronous Receiver/Transmitter
EOF	End Of File
FAT	File Allocation Table
FIFO	First In, First Out

IDE	Integrated Drive Electronics
IRQ	Interrupt Request
ISRC	International Standard Recording Code
LAN	Local Area Network
LBA	Logical Block Addressing
LED	Light-Emitting Diode
MBR	Master Boot Record
MMC	MultiMediaCard
MSF	Minute Second Frame
NMI	Non-Maskable Interrupt
NTFS	New Technology File System
PEROM	Programmable Erasable Read Only Memory
PETSCII	PET Standard Code of Information Interchange
PMA	Program Memory Area
RAM	Random Access Memory
REU	RAM Expansion Unit
ROM	Read-Only Memory
RTC	Real-time clock
SCPU	SuperCPU
SCSI-MMC	SCSI MultiMedia Commands
SD	Secure Digital
SDHC	Secure Digital High Capacity

SID	Sound Interface Device
TOC	Table Of Contents
UART	Universal Asynchronous Receiver/Transmitter
UDF	Universal Disk Format
UDP	User Datagram Protocol
UPC	Universal Product Code
USB	Universal Serial Bus
VIC II	Video Interface Chip II

List of Tables

1 C128 extra keys 26
2 Detailed directory filetypes 46
3 Bits of TOC format 71
4 Bits of sub-channel read format 72
5 Sub-channel read modes 72
6 Sub-channel read data header 72
7 Sub-channel read audio status codes 73
8 Sub-channel CD current position data format 73
9 Sub-channel control field of CD current position 74
10 Manager input window keys 77
11 Manager keys 87
12 Monitor editor keys 111
13 Monitor commands 112
14 LL list format 126
15 Default BASIC function and editor keys 131
16 Device status ($90) 134
17 Messages ($9D) 135
18 File numbers ($B8) 136
19 Device numbers ($BA) 136
20 Secondary addresses ($B9) 137
21 Error codes returned by IDEDOS and KERNAL 150
22 Direct KERNAL call replacement table 163
23 X1541 PCLink cable 171
24 Parallel PCLink cable 172
25 Serial PCLink cable 173
26 Ethernet crosslink PCLink cable 174
27 G-P data format 182

28 T-RB and T-RD data format 187
29 Disk format codes 188
30 Bits of fast forward and reverse mode byte 195
31 Volume control format 196
32 Output channel selection 197
33 ShortBus pinout 242
34 Clock-port pinout 246

List of Figures

1 IDE64 V3.4 cartridge 15
2 Start screen of the setup utility 24
3 Standard setup screen of the setup utility 25
4 Device number setup screen of the setup utility 29
5 ATA devices screen of the setup utility 31
6 The File Manager 78
7 The IDE64 builtin monitor 91
8 128 KiB PEROM upgrade 215
9 Location of the PGM pin on different cards 216
10 The Perom programmer utility 218
11 ShortBus female cable connector 237

List of Listings

1 Path handling example 49
2 BASIC directory list 58
3 Reading load address from file 59

4 SEQ file writing 60
5 Relative file copy 63
6 Link creation 63
7 Link content reading 64
8 Block read and positioning 69
9 Block read for identifying drive 69
10 Plugin sample source 79
11 MAN,USR generator source 85
12 Shell sample source 117
13 Function key redefinition 122
14 READST usage example 133
15 SETMSG usage example 134
16 STOP usage example 135
17 SETLFS, SETNAM and OPEN usage example 138
18 CLOSE usage example 139
19 CHKIN usage example 140
20 CHKOUT usage example 141
21 CHRIN usage example 142
22 GETIN usage example 143
23 GETIN usage for keyboard read 143
24 CHROUT usage example 144
25 CLALL usage example 145
26 CLRCHN usage example 146
27 LOAD usage example 147
28 SAVE usage example 149
29 IDE64 card detection 151
30 ActionReplay card detection 152
31 WRITE compatibility fallback routine 153
32 READ and WRITE usage for copy 155

33 READ compatibility fallback routine 157
34 Interrupt vector restoring 161
35 IRQ vector restoring 161
36 Vector save and restore 162
37 Error channel reading 162
38 Directory list in assembly 164
39 Relative file seek 176
40 Normal file seek 176
41 Get current partition 182
42 Check if partition exists 182
43 Disk change detection 184
44 Get power state 185
45 Channel reversing 196
46 Mono mixing 196
47 Volume query 197
48 Medium type detection 198
49 Drive capabilities detection 199

Index

accu, 19
    charging, 27
append, 61

backtrace, 101
battery, 19
binary, 100
block
    read, 67
    write, 68
boot file, 24
bugs, 13

C128, 210
    keyboard, 26, 89
CD, 119
    DOS wedge, 117
CD-ROM
    audio, 193
    commands, 193
    format, 35
    slowdown, 32
    volume, 196
CDCLOSE, 120
CDOPEN, 120
CFSfdisk, 36
CFSfsck, 221
CHANGE, 120
CHKIN, 140
CHKOUT, 141
CHRIN, 142
CHROUT, 144
CLALL, 145
clock
    read, 187
    set, 23
clock-port, 243
CLOSE, 65
    KERNAL, 139
CLRCHN, 146
CMD, 66
    emulation, 27
colors, 28
copy, 75
create
    directory, 128, 192
    partition, 37

DATE, 121
    set, 23
DE32, 21
DEF, 122

device number, 29
    change, 120, 183
DigiMAX, 239
DIR, 122
    DOS wedge, 114
direct channel, 67
    open, 58
directory, 45
    change, 119, 190
    create, 128, 192
    formatted, 58
    header, 192
    list, 45, 122
    raw, 52, 58
    remove, 128, 192
diskchange, 184
Distributors, 247
Dolphin DOS, 211
DOS wedge, 113
    disable, 26
drive capabilities, 199
DUART, 238
    PCLink, 173
DVD
    format, 35

eject, 186
error messages, 203
ETFE, 239
ETH64, 237
ETH64II, 243
Ethernet
    PCLink, 173

fastloader, 24
files, 53
filesystem
    check, 221
    format, 35
format
    disk, 188
    partition, 35
freeze, 94, 95
    point, 109
function keys
    BASIC, 131
    disable, 27
    manager, 87
    monitor, 111
    redefine, 122

GET#, 65
GETIN, 143

HDINIT, 123
header, 192
hide
    file, 180

hole, 53, 176

identify
    drive, 184
information, 247
INIT, 123
initialize, 177
INPUT#, 65
Internet, 2, 247

JiffyDOS, 211

KILL, 124
KILLNEW, 124

link, 63
LL, 125
LOAD, 56, 127
    DOS wedge, 114–116
    error, 33
    KERNAL, 147
    monitor, 93
lock
    directory, 180
    file, 180
    medium, 186
LS-120, 18
    bug, 33

MAN, 127
manager, 75
    keys, 77, 87
    start, 127
medium type, 198
memory
    read, 201
    write, 201
MKDIR, 128
modify, 61
monitor, 89
    commands, 112
    keys, 111
move, 179
MP3@64, 245

OPEN, 57
    KERNAL, 138

partition
    change, 181
    edit, 39
    hide, 40
    info, 182
    list, 43
path, 48
PCLink, 171
peripherals, 18
    clock-port, 243
    ShortBus, 237

plugin, 78
    config, 84
    format, 79
position, 175
power
    management, 31, 185
    supply, 16
    up, 19
PRINT#, 65

READ, 155
read error, 203
READST, 133
relative file, 61
remove
    directory, 128, 192
    file, 177
    partition, 37
rename
    disklabel, 38
    file, 179
    header, 192
    partition, 37
reset, 15
    drive, 184
RM, 128
RMDIR, 128
RR-Net, 244

SAVE, 55, 129
    DOS wedge, 116
    KERNAL, 149
    monitor, 93
scratch, 128, 177
seeking, 175
selftest, 20
SETLFS, 136
SETMSG, 134
SETNAM, 137
setup, 23
shell, 117
ShortBus, 237
SilverSurfer, 244
spin down, 185
sprite, 100
status, 134
STOP, 135
SuperCPU, 20, 209
    monitor, 90
SYS, 129
system drive, 29

TI$, 25

USB
    PCLink, 174

validate, 201

filesystem check, 221
VERIFY, 56, 130
DOS wedge, 115
KERNAL, 147
monitor, 94

wildcard
file, 50
monitor, 105
WRITE, 152
write protect
directory, 180
drive, 189
file, 180
partition, 39

X1541, 171

Zip drive, 17